Omega ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

A general modeling approach to online optimization with lookahead ☆

Fabian Dunke *, Stefan Nickel

Karlsruhe Institute of Technology, Institute of Operations Research, Karlsruhe, Germany

ARTICLE INFO

Article history:
Received 16 October 2014
Accepted 14 October 2015

Keywords:
Online optimization
Lookahead
Discrete event system
Algorithm analysis

ABSTRACT

A vast number of real world problems are coined by an information release over time and the related need for repetitive decision making over time. Optimization problems arising in this context are called online since decisions have to be made although not all data is known. Due to technological advances, algorithms may also resort to a limited preview (lookahead) on future events. We first embed the paradigm of online optimization with lookahead into the theory of optimization and develop a concise understanding of lookahead. We further find that the effect of lookahead can be decomposed into an informational and a processual component. Based on analogies to discrete event systems, we then formulate a generic modeling framework for online optimization with lookahead and derive a classification scheme which facilitates a thorough categorization of different lookahead concepts. After an assessment of performance measurement approaches with relevance to practical needs, we conduct a series of computational experiments which illustrate how the general concept of lookahead applies to specific instantiations and how a knowledge pool on lookahead effects in applications can be built up using the general classification scheme.

1. Introduction

Although there is an agreement on the importance of coping with unexpected events in today's systems for production and logistics [26,51], recent implementations of planning and scheduling systems still suffer from their deficiency in dealing with uncertainty over time. In a rolling horizon, plans are determined on the basis of forecasts by offline optimization methods [51]. However, since only decisions of the next period are implemented before the problem gets resolved with updated data, this approach exhibits large redundancies.

On the other hand, possibilities for collection of data about near-future events are steadily increasing due to technological developments [26] such as radio frequency identification (RFID), global positioning systems (GPS) or geographical information systems (GIS). Since planning systems in these environments are subject to permanent information inflow, they are said to be online. Optimization problems in this context are called online optimization problems [24]. These problems are characterized by the fact that decisions are required to be made repeatedly before all data is available. In contrast to other methodologies for optimization under uncertainty, there are no forecasts or probabilities of future events assumed in online optimization. However, as a result of technological opportunities given above, we can now cope with uncertainty differently. Through the installation of lookahead devices, it is possible to acquire data about future events at an earlier point in time. Hence, uncertainty is tackled forcefully because parts of the previously uncertain future can now be fixed to certainty through the utilization of lookahead. Thus, the decision making process consists of repetitive decisions where the input to each decision only consists of the small, but certain part of the future known at that time. Though, as can be seen from the different information gathering devices mentioned above, it may be reasonable to be more precise with respect to the actual degree of "onlineness" in a specific problem setting. The need for a concise notion of lookahead is also reflected by the manifold perceptions of lookahead depending on the application [2–4,14,17,29,37,45,52,56]. For this reason, this paper coins the notion of online optimization with lookahead on a formal basis.

The task of solving online optimization problems is a recurring pattern in industrial applications (Fig. 1): each time the functional logic of a dynamic system requires a decision, an online algorithm is called to deliver it, i.e., partial answers based on currently available data have to be given such that the overall solution will be as good as possible.

Solution methodologies for the different optimization paradigms strongly differ from each other. Consider the input sequence

☆This manuscript was processed by Associate Editor Ghate.
* Corresponding author.
E-mail addresses: fabian.dunke@kit.edu (F. Dunke), stefan.nickel@kit.edu (S. Nickel).

http://dx.doi.org/10.1016/j.omega.2015.10.009
0305-0483/

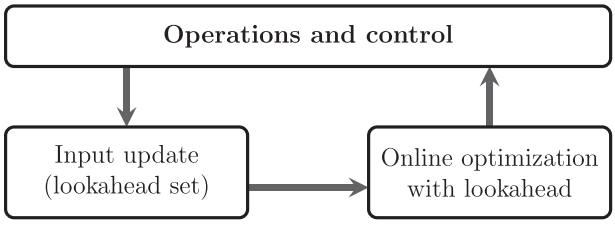

Fig. 1. Hierarchical relation between operations and control of a dynamic system and online optimization with lookahead.

$\sigma = (\sigma_1, \sigma_2, \ldots)$. In offline optimization, σ is known in advance and a plan for how to process its elements can be computed directly. In the sequential model of online optimization [30], only one input element is known at a time and input elements must be processed in release order, i.e., input element σ_i is processed based upon knowledge of $\sigma_1, \ldots, \sigma_i$ and previous decisions on $\sigma_1, \ldots, \sigma_{i-1}$. In the time-stamp model [30], each input element is assigned an arrival date such that input elements may accumulate naturally and automatically form some lookahead set of unprocessed input elements. In online optimization with request lookahead [2,52], more than one unprocessed input element may be known at a time and also an explicit formulation of processing restrictions is required. One could insist on sequential processing (σ_i must be processed before σ_{i+1}) or allow for a processing in arbitrary order (σ_{i+1} can be processed before σ_i). There are a plenty of variations of how lookahead is understood and what it means for the processing of single input elements. For this reason, this paper will provide the tools for classifying the main features of a specific online optimization problem with lookahead.

Literature focuses on a worst-case type analysis of algorithm performance – called competitive analysis [13,40] – where an online algorithm has to compete with an optimal offline algorithm. Derivations of this measure are based on the individual taxonomy in a specific problem and not on a general notation valid for all problems. Likewise, the case with lookahead has been addressed only rarely in specific problems from routing and transportation [4–6,36,37,52], scheduling [18,44–47,54,57,58], organization of data structures [2,3,14,41,53,55,56], data transfer [20,34], packing [29,31], lot sizing [1], metrical task systems [9,42] or graph theory [17,32,35]. To the best of our knowledge, there have been no attempts to formalize different degrees of available information in a general framework. A reason for the lack of general concepts lies in the different possibilities to deal with temporal aspects [30]. When only the order of input element releases matters, time is already modeled implicitly through the indices of $\sigma_1, \sigma_2, \ldots$ On the other hand, when time durations play a role, time aspects have to be modeled explicitly as part of the data belonging to $\sigma_1, \sigma_2, \ldots$, e.g., in the form of release times $t_1, t_2, \ldots$ This issue also accounts for various perceptions of lookahead along with its implied processing characteristics.

1.1. Lookahead and related concepts for uncertainty

The idea of online optimization with lookahead is based on known deterministic information previews, and hence it can be distinguished from other approaches for optimization under incomplete information or uncertainty. In stochastic programming [12], probability distributions for scenarios that take into account all uncertain factors (often in form of parameters) are known and solution quality is typically evaluated by average-case measures to immunize the solution probabilistically to incomplete information. In addition, stochastic programming is rather concerned with sporadic than with frequent decision making. Scenarios are often coined by the realization of parameter values which are considered to be some random variable. Online optimization differs from this approach strongly since it is not focused on scenarios and/or parameters, but rather on the realizations of input elements for which no stochastic principles are known to hold. Dynamic programming [8] assumes that an optimization problem exhibits the property of optimal substructures (i.e., an optimal solution is composed of optimal solutions to subproblems) and the property of overlapping subproblems (i.e., the overall problem can be broken down into several subproblems of the same type whose solutions can be composed to obtain an overall solution). Clearly, both properties are not fulfilled in general in the online versions of combinatorial optimization problems. Moreover, many online problems involve time considerations such as input element release times. Therefore, it is impossible to subdivide the overall problem into several discrete stages. To apply the theory of dynamic programming, we therefore need a fixed time horizon in order to determine an optimal solution. In online optimization, the end of the input sequence is not known and decisions are made in an exclusively forward-moving rolling time horizon. The time horizon T in dynamic programming is the number of periods for which the planning shall be conducted, and between periods 1 and T all possible realizations in the respective periods are considered. Conceptually different, lookahead in online optimization only takes into account the actual upcoming realizations (known due to some lookahead device) and not all possible realizations. Finally, the goal in dynamic programming is different than in online optimization: in dynamic programming we are looking for an optimal strategy for given horizon, state space, action space, state transition and reward function which all are known in advance; in online optimization, we usually already have a strategy in form of an algorithm and want to check its behavior in the online setting. Nonetheless, it is possible to emulate the behavior of an online algorithm by means of a Markov chain (cf. also [22]). However, this approach is very unhandy and leads to computational issues even for small problem instances. We also note that the setup of a Markov decision process [49] is different from online optimization: state transitions occur probabilistically once a control action has been chosen, whereas in our setting they occur deterministically based on an algorithm's deterministic decision. Markov decision processes are used as a modeling formalism to determine an optimal strategy, i.e., the decisions of an optimal algorithm with respect to some expected objective value, using dynamic programming. Stochastic assumptions concerning transition probabilities depending on the control action are given a priori: $p(s, a, s')$ with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ is the probability that the successor state of s is s' if action a is chosen. In contrast to this, our analysis merely intends to evaluate the quality of a given algorithm in a setting of complete nescience of stochastic information. In particular, it is not possible to find an optimal algorithm because the end of the horizon is unknown. The field of model predictive control [15] deals with finding the optimal control of complex dynamic systems. This idea is similar to that of online optimization with lookahead. However, the setting in model predictive control is relatively clear marked out by relations between dependent and independent variables in a corresponding process model. This is also why this technique is mainly used in the context of process industries, but not in the field of combinatorial online optimization. Robust optimization [11] does not rely on probability distributions but on a given range of possible values for uncertain factors. The goal is to construct a solution which is feasible for all possible realizations and exhibits optimality in some robustness-related sense. In online optimization with lookahead, there is no need for forecasts and probabilities, and subproblems are computationally easy because of their limited size. From this discussion we see that the concept of "lookahead" is seen from quite a number of different perspectives.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

In this paper, lookahead is understood in a deterministic way, i.e., it can be taken for granted that the information in the lookahead is always correct and never wrong. This is also the perception of lookahead in the online optimization community (cf. [2–4,14,17,29,37,45,52,56]) since it is presumed that there are no probabilistic information given in many applications. As a result, it could be even more harmful to introduce wrong hypotheses about underlying probability distributions when they are actually unknown. Quite the contrary, online optimization traditionally refrains from introducing probabilistic models about future events in order to hedge against all futures in equal measure. In this sense, lookahead in online optimization provides algorithms with additional power in the form of certain information about the future.

1.2. Problem statement

In industrial practice, we encounter many applications which contain online optimization problems, e.g., in machine scheduling [48] or vehicle routing [28]. On the other hand, no general framework or tool set for a comprehensive analysis of online optimization algorithms in applications is available yet. Moreover, there is not even a unique definition of lookahead. Hence, we first need a common understanding of lookahead in order to understand how an algorithm reacts upon additional data and how it could hopefully be further improved. Following this approach will make it possible to trace back lookahead effects observed in practice to their primary origins and ascribe them a core reason. Moreover, lookahead mechanisms and impacts occurring throughout different domains can then be described, explained and compared using a general notation.

The remainder of this paper is organized as follows: Section 2 presents a general definition of the concept of lookahead in optimization and provides a detailed specification of the mechanisms by which additional value in terms of improved algorithm performance can be gained from additional information. In Section 3, we formulate a general modeling framework which allows us to model the solution process in an instance of an online optimization problem. The resulting framework also leads to the introduction of a classification scheme by which the key components of a specific lookahead concept can be categorized quickly. After a short assessment of performance measures in Section 4, the computational experiments in Section 5 show how lookahead works in different problem classes and how the results can be used to build a pool of knowledge on lookahead effects.

2. Online optimization with lookahead

In order to establish a clear, but yet flexible definition for an online optimization problem with lookahead, we first introduce some basic concepts with respect to the process of information release in an optimization problem and related processing possibilities of an algorithm.

2.1. Definition of lookahead

According to Ausiello et al. [7], a (single-objective) optimization problem Π is a quadruple (I,S,f,opt) where I is a set of instances, S is a function returning the set of solutions $S(i)$ for any $i\in I$, f is a function returning the objective value for any pair $(i,s)\in I\times S(i)$, and $\text{opt}\in\{\min,\max\}$ is the optimization goal. However, this definition does not account for the sequentiality in the instance revelation process that any online solution method has to obey, and it disregards previous answers of the solution method. We introduce the instance revelation rule as a mechanism to account for dynamic aspects in the revelation process of an instance. Essentially, this comprises a description of when input elements are released over time.

Definition 1 (*Instance revelation rule*). An instance revelation rule is a rule that governs the temporal course of events in the release of information on the problem instance.

The close link between dynamic input disclosure and a corresponding instance revelation rule is respected in the following definition which is an extension of the rather general definition for an instance of an optimization problem given by Garey and Johnson [25]. By including the instance revelation rule into the definition of a problem instance we explicitly account for the dynamic character of the online optimization paradigm.

Definition 2 (*Instance of an optimization problem*). An instance of an optimization problem consists of a set of parameter values including an input sequence $\sigma=(\sigma_1,\sigma_2,\ldots)$ and an instance revelation rule r.

From additional data, new possibilities for possible actions of an algorithm may arise. Hence, we need a way to settle all unclarities concerning the processing of the input elements which may be implied by additional data. Hence, we associate a set of rules with a problem.

Definition 3 (*Rule set*). A rule set of a problem is a set of restrictions on the solution to an instance of the problem.

Now, we can associate each instance $i\in I$ with a specific instance revelation rule that determines how the information of i is made available and with a rule set that affects the form of the set of feasible solutions $S(i)$ for each $i\in I$. We note that the rule set P gives us a description of the conditions under which an algorithm has to make its decisions during input element processing, whereas the feasible set S will only be known after all input elements have been released. Of course, each element in S has to obey the rules given in P. In order to prepare for the definition of lookahead, we make the following notational conventions: when we substitute instance revelation rule r by instance revelation rule r', then we write $r\to r'$ and speak of an instance revelation rule substitution. When we substitute rule set P by rule set P', then we write $P\to P'$ and speak of a rule set substitution. Definition 4 suggests that lookahead consists of an informational component $(r\to r')$ and a processual component $(P\to P')$. The subdivision of lookahead into these two components will be helpful to understand and explain the behavior of an algorithm in an online optimization problem.

Definition 4 (*Lookahead*). A lookahead is a pair $(r\to r',P\to P')$ consisting of an instance revelation rule substitution $r\to r'$ and a rule set substitution $P\to P'$.

Accordingly, an online optimization problem with lookahead (I',S',f,opt) can be seen as the result of applying the two components of lookahead to a reference online optimization problem (I,S,f,opt). In the first step, information is made available earlier which is achieved through the instance revelation rule substitution $r\to r'$. In the second step, potential new processing possibilities are established through the rule set substitution $P\to P'$. The first step modifies the instance set from I to I'. The second step modifies the feasible set S to S' because a changed rule set also changes what we consider to be a feasible solution. Note that we could also artificially bypass the second step (i.e., $P\to P$). This would give us an online optimization problem with lookahead where all benefits of lookahead are exclusively due to the earlier information release.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

2.2. Value of lookahead information

Due to the lack of a neutral performance benchmark in optimization under incomplete information, we cannot derive any statement about the value of lookahead without relating it to a specific algorithm. Hence, we ask what can be achieved by an algorithm upon provision of additional lookahead. The discussion is restricted to minimization problems.

Definition 5 (*Lookahead value*). Let $\Pi_P=(I,S,f,\min)$ and $\Pi_{P'}=(I',S',f,\min)$ be optimization problems where $i'\in I'$ results from applying lookahead $(r\to r',P\to P')$ to $i\in I$, and let $s_{\mathrm{ALG}}\in S(i)$ and $s_{\mathrm{ALG}'}\in S'(i')$ be the solutions determined by algorithm ALG for instance i in problem Π_P and by algorithm ALG′ for instance i' in problem $\Pi_{P'}$, respectively. The lookahead value of $(r\to r',P\to P')$ on i with respect to $(\mathrm{ALG},\mathrm{ALG}')$ is

$$\Delta f^{r,r',P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i):=f(i,s_{\mathrm{ALG}})-f(i',s_{\mathrm{ALG}'}). \tag{1}$$

The following decomposition of $\Delta f^{r,r',P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i)$ is artificial because it relies on the members of the set $\mathcal{ALG}$ of admissible algorithms for problems that have to operate under rule set P.

Definition 6 (*Partial lookahead value due to $r\to r'$*). Let $\Pi_P=(I,S,f,\min)$ be an optimization problem, let ALG be an algorithm for Π_P, and let $\mathcal{ALG}$ be a set of admissible algorithms for Π_P. Further, let i'' be the instance which results from applying lookahead $(r\to r',P\to P')$ to $i\in I$, and let $s_{\mathrm{ALG}}\in S(i)$ and $s_{\mathrm{ALG}''}\in S(i'')$ be the solutions determined by algorithm ALG for instance i and by algorithm $\mathrm{ALG}''\in\mathcal{ALG}$ for instance i'', respectively. The partial lookahead value of $(r\to r',P\to P')$ due to instance revelation rule substitution $r\to r'$ on i with respect to $(\mathrm{ALG},\mathcal{ALG})$ is

$$\Delta f^{r,r'}_{\mathrm{ALG}}(i):=f(i,s_{\mathrm{ALG}})-\min_{\mathrm{ALG}''\in\mathcal{ALG}}\left\{f(i'',s_{\mathrm{ALG}''})\right\}. \tag{2}$$

In this definition, ALG″ has to operate under P although the lookahead also comprises a rule set substitution to P'. However, to determine the partial lookahead value attributable to the instance revelation rule substitution, we have to maintain processing under P.

Definition 7 (*Partial lookahead value due to $P\to P'$*). Let $\Pi_P=(I,S,f,\min)$ and $\Pi_{P'}=(I',S',f,\min)$ be optimization problems, let ALG and ALG′ be algorithms for Π_P and $\Pi_{P'}$, respectively. Further, let i' be the instance which results from applying instance revelation rule substitution $(r\to r')$ to $i\in I$, and let $s_{\mathrm{ALG}'}\in S(i')$ be the solution determined by algorithm ALG′ for instance i'. The partial lookahead value of $(r\to r',P\to P')$ due to rule set substitution $P\to P'$ on i with respect to $(\mathrm{ALG},\mathrm{ALG}')$ is

$$\Delta f^{P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i):=\Delta f^{r,r',P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i)-\Delta f^{r,r'}_{\mathrm{ALG}}(i). \tag{3}$$

By definition, the lookahead value is decomposed such that for lookahead $(r\to r',P\to P')$ it holds that

$$\Delta f^{r,r',P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i)=\Delta f^{r,r'}_{\mathrm{ALG}}(i)+\Delta f^{P,P'}_{\mathrm{ALG},\mathrm{ALG}'}(i). \tag{4}$$

Note that the partial lookahead value due to rule set substitution implicitly presumes that additional information is made known earlier. Hence, it corresponds to the part of the lookahead value that could not be elicited just from the additional information. The following examples show that – depending on the application – both types of partial lookahead values can be prevalent.

Example 1 (*Online bin packing with lookahead*). The task in (one-dimensional) bin packing is to pack a number of items $\sigma_1,\sigma_2,\ldots,\sigma_n$ for $n\in\mathbb{N}$ with sizes $s_i\in(0,1]$ for $i=1,\ldots n$ into a minimum number of unit-capacitated bins. We now only consider two item sizes $\{0.4,0.6\}$ and $\sigma=(\sigma_1,\sigma_2,\sigma_3,\sigma_4)=(0.4,0.4,0.6,0.6)$. For the pure online version, we have instance revelation rule r and rule set P as
r:=σ_1 is known at the beginning; σ_{i+1} is revealed after σ_i has been assigned,
P:={σ_i has to be assigned before σ_{i+1}}.
For the lookahead version, we assume request lookahead of size 2, i.e., we have instance revelation rule r' and rule set P' as r':=σ_1,σ_2 are known at time 0; an item is revealed when another one has been assigned,
P':={Any previously revealed item can be assigned when it has not yet been assigned}.

Let instances i and i' correspond to σ as revealed under r and r', respectively. We use algorithm BestFit (BF) in the pure online case and algorithm BestFitDecreasing (Bfd) in the lookahead case. BF puts the next item into the fullest bin available, whereas Bfd first sorts the known unassigned items by non-increasing sizes and assigns the largest of them to the fullest bin available. Both algorithms comply with rule sets P and P', respectively. Moreover, we let intermediary algorithm BestFitModified (Bfm) operate under P, i.e., it puts the items into the bins in their order of appearance. However, Bfm bases its decision on all available information: when Bfm is supplied with information according to r' it operates identically to BF except for the case where an open bin at level 0.4 exists and two items with sizes 0.4 and 0.6 have to be assigned in this order (because of P). In this case, the item of size 0.4 is put in a new bin and the item of size 0.6 is packed in a bin at level 0.4.

BF yields one bin at level 0.8 and two bins at level 0.6; Bfd yields two full bins. Hence, $\Delta f^{r,r',P,P'}_{\mathrm{BF},\mathrm{Bfd}}(i)=3-2=1$. Bfm also yields two bins. Moreover, $f(i',s)\geq 2$ for $s\in S(i')$. Thus, $\Delta f^{r,r'}_{\mathrm{BF}}(i)=3-2=1$ and $\Delta f^{P,P'}_{\mathrm{BF},\mathrm{Bfd}}(i)=3-2-1=0$, i.e., the improvement is a result of provision of information at an earlier point in time (instance revelation rule substitution), allowing us to permute the items (rule set substitution) has no value.

Example 2 (*Online TSP with lookahead*). Given a metric space $\mathcal{M}$ with distance function d, the task in the traveling salesman problem (TSP) is to find a round trip (tour) for the locations (requests) $\sigma_1,\sigma_2,\ldots,\sigma_n$ for $n\in\mathbb{N}$ with points $x_i\in\mathcal{M}$ for $i=1,\ldots n$ to be visited such that some cost depending on the total travel distance is minimized. We now only consider two locations $\{0,1\}$ and $\sigma=(\sigma_1,\sigma_2,\sigma_3,\sigma_4,\sigma_5)=(0,1,0,1,0)$. The server starts in 0. For the pure online version, we have instance revelation rule r and rule set P as
r:=σ_1 is known at the beginning; σ_{i+1} is revealed after σ_i has been visited,
P:={σ_i has to be visited before σ_{i+1}}.

For the lookahead version, we assume request lookahead of size 2, i.e., we have instance revelation rule r' and rule set P' as
r':=σ_1,σ_2 are known initially; a request is revealed when another one has been visited,
P':={Any previously revealed request canbevisited when it has not yet been visited}.

Let instances i and i' correspond to σ as revealed under r and r', respectively. P' makes locations visitable earlier, e.g., at time 0, both σ_1 and σ_2 can be visited, while under P only σ_1 can be visited. Due to P, we must use algorithm FirstComeFirstServed (Fcfs) in the pure online setting; in the setting with lookahead, we use algorithm NearestNeighbor (Nn). Fcfs visits the requests in their order of release, whereas Nn stays in the current location when this location is contained in the two unvisited known requests. Both algorithms comply with rule sets P and P', respectively. Moreover, there is no alternative to Fcfs under P, i.e., we must choose Fcfs as

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

an intermediary algorithm that is supplied with information according to r', but obviously cannot capitalize from it under preservation of P.

FCFS yields a distance of 4; NN yields a distance of 2. Correspondingly, $\Delta f^{r,r',P,P'}_{\text{FCFS,NN}}(i) = 4-2 = 2$. Applying "intermediary" algorithm FCFS trivially yields a distance of 4 in compliance with P. Moreover $f(i', s) = 4$ for $s \in S(i')$. Thus, $\Delta f^{r,r'}_{\text{FCFS}}(i) = 4-4 = 0$ and $\Delta f^{P,P'}_{\text{FCFS,NN}}(i) = 4-2-0 = 2$, i.e., the improvement is a mere result of the change of circumstances under which requests have to be visited (rule set substitution), allowing us to permute the order of the requests is responsible for the complete lookahead value. Forwarded information (instance revelation rule substitution) has no value when it is separated from the rule set substitution.

2.3. Relation to discrete event systems

So far, we only paid attention to the outcome of an algorithm on a given problem instance but not to the solution process. Interpreting the release of an input element as an event, we recast the solution process in online optimization using discrete event systems terminology.

Define a process as a transformation, transportation or storage of an energetic, material or informational resource, then a system is a fragment of the real world in which all processes relevant to the user take place. Hence, a system is a reduction of the real world to those entities which influence these processes and satisfy the user's wish to duplicate them [16]. Typically, a system reacts to a change in the input values with an associated change in the output values. In this context, let an event be a spontaneous occurrence triggered by an external entity or by fulfilling the conditions of a switching rule which is capable of submitting changes of the input to a system, then a discrete event system is a system whose state trajectory only depends on the occurrence of discrete events over time [16].

In online optimization, the release of input elements occurs at discrete points in time and any online algorithm operates as part of a reactive planning system. Hence, the solution process can be modeled as a discrete event system. Table 1 summarizes corresponding analogies. As a result of the similarities, online algorithms and discrete event systems can be tackled with the same modeling tools such as automata, Markov chains and discrete event simulation. However, due to the focus on optimization and lookahead, specializations and adjustments are needed to obtain a general framework for online optimization with lookahead.

3. Modeling framework and classification scheme

This section focuses on the modeling of the solution process in online optimization with lookahead by means of a general framework. Based upon the discrete event process model that describes the interaction of the modeling elements over time, a classification scheme is derived which facilitates a general categorization of different lookahead concepts.

Table 1
Analogies between online optimization and discrete event systems.

Online optimization	Discrete event system
Input element	Event
Input sequence	Event sequence
Input element release	Event occurrence
Releases occur at discrete times	Events occur at discrete times
Algorithm reacts to input element release	System reacts to event occurrence
Online algorithm	State transition function
Objective value	Entry in state encoding scheme

3.1. Previous modeling approaches

A first attempt to model online optimization generically was request answer games [10]. A sequence of requests is presented online and each time a new request arrives, an answer has to be given incurring some cost. Formally, a request answer game over time horizon $n \in \mathbb{N}$ is a triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ where $\mathcal{R}$ is a set of requests, $\mathcal{A}$ is a set of answers and $\mathcal{C} = \{c_n \mid n \in \mathbb{N}\}$ is a set of cost functions with $c_n : \mathcal{R}^n \times \mathcal{A}^n \to \mathbb{R} \cup \{\infty\}$. A deterministic online algorithm $\text{ALG} = (\text{ALG}_1, \text{ALG}_2, \ldots, \text{ALG}_n)$ is a sequence of functions with $\text{ALG}_i : \mathcal{R}^i \to \mathcal{A}$ for $i = 1, 2, \ldots, n$. The output of an algorithm upon request sequence $r = (r_1, r_2, \ldots, r_n) \in \mathcal{R}^n$ is a sequence of answers $a = (a_1, a_2, \ldots, a_n) \in \mathcal{A}^n$ with $a_i = \text{ALG}_i(r_1, r_2, \ldots, r_i)$ for $i = 1, 2, \ldots, n$. The costs of ALG on r are $c_n(r, a)$. The concept could not be established as a modeling standard due to its lacking ability to account for an algorithm's rationale.

Another proposition for a generic model origins from the priority programme "Online Optimization of Large Scale Systems" of the German Research Foundation [30] where the sequential model and the time stamp model are introduced. In both models, $\sigma = (\sigma_1, \sigma_2, \ldots)$ is revealed over time and an algorithm has to serve each of the input elements. In the sequential model, σ_{i+1} is presented only when σ_i has just been served. In the time stamp model, σ_i comes along with a release time τ_i which is independent of an algorithm's previous actions. The release time represents the earliest time where σ_i may be served. Hence, opposite to the sequential model, the order of service is not fixed initially. Under the time stamp model, decisions have a tentative character as long as they have not been executed and may be revoked until then. For both models, no extension to lookahead was suggested.

3.2. Modeling framework components

Since previous models of online optimization lack an integration of lookahead and a representation of the sequential solution process, we build a general modeling framework for online optimization with lookahead based on the following building blocks:

- The basic modeling elements provide an abstract view on the infrastructure of information flows needed to model the solution process.
- The lookahead type specifies the instance revelation rule that an algorithm under lookahead has to obey in contrast to the reference online case.
- The processing characteristics specify the processing rules that an algorithm under lookahead has to obey in its decision making on how to serve the input elements.
- The algorithm execution mode defines the time instants at which algorithm evaluation is scheduled over the course of the solution process.

3.2.1. Basic modeling elements

We first identify the constituting elements which are needed in each model of online optimization with lookahead. To this end, we consider the problem as part of a dynamic system which consists of the collection of all real world entities needed to describe the problem.

Input element and input sequence: A piece of information is given by an input element from the set of all possible input elements Σ_1 (which is the set of all input sequences of length 1). The input appears in form of an input sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$ with $\sigma_i \in \Sigma_1$ for $i = 1, 2, \ldots$ The elements of σ are revealed successively according to the lookahead mechanism. Each input element awaits some processing during the solution process. Formally, an input element σ_i is a data record containing all information needed by an algorithm to decide on the processing of σ_i:

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

- Release time $\tau_i := \tau(\sigma_i)$ in the reference online case; release time $\tau'_i := \tau'(\sigma_i)$ in the lookahead case with $\tau'_i \le \tau_i$.
- Processing time interval $[\underline{T}_i, \overline{T}_i] := [\underline{T}(\sigma_i), \overline{T}(\sigma_i)]$ in the reference online case; processing time interval $[\underline{T}'_i, \overline{T}'_i] := [\underline{T}'(\sigma_i), \overline{T}'(\sigma_i)]$ in the lookahead case.
- Input element information $r_i := r(\sigma_i)$.

$[\underline{T}_i, \overline{T}_i]$ and $[\underline{T}'_i, \overline{T}'_i]$ give the times at which σ_i is allowed to be (physically) processed in the absence and presence of lookahead, respectively. r_i carries the essential information of σ_i which is needed for decision making (e.g., the size of an item in bin packing, location of a request in the TSP, job processing times in scheduling, or input element execution times).

Example 3 (*Input element data*). Consider a manual order picking system with three (initially unknown) picking requests $\sigma_1, \sigma_2, \sigma_3$ and a shift from 09:00 until 18:00. In the pure online setting, we may have release times $\tau_1 = 09{:}00$, $\tau_2 = 09{:}05$, $\tau_3 = 09{:}08$. Hence, processing time intervals are $[\underline{T}_1, \overline{T}_1] = [09:00, 18:00]$, $[\underline{T}_2, \overline{T}_2] = [09:05, 18:00]$, $[\underline{T}_3, \overline{T}_3] = [09:08, 18:00]$. By applying time lookahead of five minutes, we will get $\tau'_1 = 09:00$, $\tau'_2 = 09:00$, $\tau'_3 = 09:03$ and $[\underline{T}'_1, \overline{T}'_1] = [09:00, 18:00]$, $[\underline{T}'_2, \overline{T}'_2] = [09:00, 18:00]$, $[\underline{T}'_3, \overline{T}'_3] = [09:03, 18:00]$. Since in this example there are no time window specifications or service level constraints for the picking requests, the lookahead automatically leads to changes in the processing time intervals. Additionally, r_i for $i = 1, 2, 3$ carries the warehouse location of the picking requests.

We associate two time-dependent variables with σ_i:

- Processing status $p_i(t) := p(\sigma_i, t) \in \{\text{unprocessed}, \text{processing}, \text{finished}\}$.
- Action $a_i(t) := a(\sigma_i, t)$.

$p_i(t)$ gives the state of σ_i at time t and rules membership to the sets of still unprocessed, currently processing or already finished input elements. $a_i(t)$ contains all information concerning how σ_i is, was, or is planned to be processed. $a_i(t)$ is determined by an algorithm; if no action has been determined yet at time t, we set $a_i(t) = \text{NULL}$. When $a_i(t) \neq \text{NULL}$, then $a_i(t)$ contains at least the elements $t_i^{a,start}$ and $t_i^{a,finish}$ denoting the times when processing of σ_i is (planned to be) started and finished, respectively. Decisions may be revoked until time $t_i^{a,start}$. The set of all possible actions for an input element is denoted by $\mathcal{A}$. We note that each input element experiences two steps: first, its information is announced. Second, this information is exploited by an algorithm to determine how the input element will be processed. Processing then takes place within $[t_i^{a,start}, t_i^{a,finish})$ without any further ado when time $t = t_i^{a,start}$ is reached and action $a_i(t_i^{a,start})$ still prescribes action starting time $t_i^{a,start}$.

Lookahead set: At each time t, the processing statuses of the known input elements establish a partition into the sets U_t, P_t and F_t of unprocessed, currently processing and finished input elements, respectively. In the reference online case, we have

$$U_t := \{\sigma_i \mid \tau_i \le t, p_i(t) = \text{unprocessed}\}, \tag{5}$$

$$P_t := \{\sigma_i \mid \tau_i \le t, p_i(t) = \text{processing}\}, \tag{6}$$

$$F_t := \{\sigma_i \mid \tau_i \le t, p_i(t) = \text{finished}\}. \tag{7}$$

In the lookahead case, τ_i is replaced with τ'_i. The following equivalences rule membership of σ_i to these sets:

$$p_i(t) = \text{unprocessed} \Leftrightarrow a_i(t) = \text{NULL} \vee t_i^{a,start} > t, \tag{8}$$

$$p_i(t) = \text{processing} \Leftrightarrow t_i^{a,start} \le t < t_i^{a,finish}, \tag{9}$$

$$p_i(t) = \text{finished} \Leftrightarrow t_i^{a,finish} \le t. \tag{10}$$

The lookahead set L_t at time t contains those input elements which have been revealed, but still require some processing, i.e., $L_t := U_t \cup P_t$. Thus, L_t depends on previous decisions and actions caused by an algorithm. An input element can stay in L_t arbitrarily long and suffer starvation if its processing is continuously rejected in favor of another input element. From a computational point of view, it suffices in most cases only to keep track of the input elements in L_t, i.e., whenever an algorithm makes its decisions independent of any element in F_t, then there is also no potential benefit of explicitly storing F_t.

State space: For a better representation of the dependencies between an algorithm's decision and the current state, we decompose each state s from the set of all states $\mathcal{S}$ into three components $s = (s^{in}, s^{sys}, s^{obj})$ with input state s^{in}, system state s^{sys} and objective state s^{obj}. All information concerning the input of the problem at time t is collected in $s_t^{in} = (U_t, P_t, F_t)$; the set of all input states is denoted by $\mathcal{S}^{in}$. Because s_t^{in} contains σ_i with $\tau_i \le t$ or $\tau'_i \le t$, it also contains all action variables $a_i(t)$ for these σ_i. To describe external circumstances, the system state s_t^{sys} at time t contains all information on the system configuration (e.g., bin configurations in bin packing or the current server position in the TSP) at time t; the set of all system states is denoted by $\mathcal{S}^{sys}$. In an optimization context, we use valued states to keep track of the current objective value during the solution process. At time t, we extract all information relevant to the future development of the objective value (e.g., the plain current objective value) in s_t^{obj}; the set of all objective states is denoted by $\mathcal{S}^{obj}$. The set of all states is $\mathcal{S} = \mathcal{S}^{in} \times \mathcal{S}^{sys} \times \mathcal{S}^{obj}$; concerning the solution process, we are interested in the state trajectory evolution $(s_t)_{t \ge 0}$ with $s_t \in \mathcal{S}$ for $t \ge 0$.

Event space: Because an algorithm has to respond to arriving input elements, we classify the input element arrival as an event. Likewise, finished processing may change the objective state such that the processing end of an input element also represents an event. In total, all state transitions which influence the solution process are triggered by an event. We subsume all possible events in the event set $\mathcal{E}$. Events occur instantaneously at discrete times. In our framework, events are the only source of uncertainty for an algorithm. The sequential and the time stamp model only know two event types referring either to finished processing or to first notification of an input element; both types coincide in the sequential model because a new element only becomes available when a known one finishes processing.

Algorithm: Input element processing is controlled by the decisions of an algorithm ALG, causing changeovers of input elements between U_t, P_t and F_t. Since the overall solution is composed of a sequence of partial solutions, ALG successively produces partial solutions by determining the values of input element action variables. Let $n_t := |U_t \cup P_t| = |L_t|$ be the number of unfinished input elements, then ALG is a family of functions $\text{ALG} := (\text{ALG}_t)_{t \ge 0}$ where

$$\text{ALG}_t : \mathcal{S} \times \mathcal{E} \to \mathcal{A}^{n_t} \tag{11}$$

is a function determining for each of the n_t known and yet unfinished input elements an action from the action space $\mathcal{A}$ based on current state $s \in \mathcal{S}$ and occurring event $e \in \mathcal{E}$. Hence, our definition of an algorithm generalizes that of the request answer games in terms of the dependency on the current state and the multidimensionality of the codomain. ALG_t is evaluated at each time t where an event occurs.

State transition: The state transition function

$$f : \mathcal{S} \times \mathcal{E} \to \mathcal{S} \tag{12}$$

determines for a given state $s \in \mathcal{S}$ and an occurring event $e \in \mathcal{E}$ the successor state $s' \in \mathcal{S}$. It only needs to be evaluated at the discrete

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

times where an event occurs because for all other times the state trajectory is assumed to advance deterministically such that it can be precomputed.

3.2.2. Lookahead type

The lookahead type specifies the mechanism under which membership of input elements to lookahead set L_t is governed. According to the taxonomy from Section 2.1, it corresponds to the instance revelation rule. Which lookahead type is employed depends on the application and on technical possibilities. We give some frequently used lookahead types, harmonize them with our notation and introduce property lookahead as a generalization of the vast majority of lookahead types. A similar classification has been given in [52].

Request lookahead: Request lookahead of size $k \in \mathbb{N}$ [52] is defined to have access to a fixed number k of unprocessed or currently processing input elements (or to all if there are less than k of them). The first of these input elements is also known in the pure online case, but the remaining $k-1$ input elements (or all if there are less than k of them) are known due to the lookahead capability. Request lookahead construes the lookahead set dependent on the processing statuses of the input elements and not in an independent process of release. We obtain the release times recursively from

$$\tau_i := \begin{cases} 0, & \text{if } i = 1, \\ \min\{t_j^{a,finish} \mid \sigma_j \in L_{\tau_{i-1}}\}, & \text{if } i = 2,3,\ldots \end{cases} \tag{13}$$

in the case without lookahead and

$$\tau'_i := \begin{cases} 0, & \text{if } i = 1,\ldots,k, \\ \min\{t_j^{a,finish} \mid \sigma_j \in L_{\tau'_{i-1}}\}, & \text{if } i = k+1, k+2,\ldots \end{cases} \tag{14}$$

in the case with lookahead. Request lookahead mainly origins from applications where time is not modeled explicitly. Request lookahead is realistic for applications which adhere to capacities (storage devices, inventory spaces), whereas it is unrealistic for applications which deal with immaterial requests (emergency calls, customer demands).

Under request lookahead, input elements are revealed on a rolling basis. However, one can also think of situations where information is released in batches (blocks): a new batch is only given when all elements of the previous one have finished processing. This gives rise to a batched version of this lookahead variant. In batched request lookahead of size k, input elements are always revealed in blocks of k elements if there are more than k elements left, otherwise the remaining input elements are revealed. In this case, we have

$$\tau'_i := \begin{cases} 0, & \text{if } i = 1,\ldots,k, \\ \max\{t_j^{a,finish} \mid \sigma_j \in L_{\tau'_{ck}}\}, & \text{if } i = ck+1, ck+2,\ldots,ck+k \text{ with } c \in \mathbb{N}. \end{cases} \tag{15}$$

Time lookahead: Time lookahead of length $D \in \mathbb{R}^{>0}$ [52] essentially makes input element σ_i known D time units earlier than in the pure online case. Release times τ_i in the pure online case are assumed independent of algorithm processing and we obtain that

$$\tau'_i := \max\{\tau_i - D, 0\}. \tag{16}$$

Time lookahead is an artificial offset of the moment at which an input element is notified by D time units. A drawback is that arbitrarily many input elements may reside in the lookahead set such that the workload may collapse in the long run.

Collective property lookahead: Collective property lookahead has been introduced as a general concept in [52] although in bin packing [29] and paging [2] it had been instantiated before. Let $c-prop_j$ be a time variant property of a subset $\sigma_{\leq j}(t)$ of the elements in input sequence σ where $\sigma_{\leq j}(t) := \{\sigma_i \mid i \in \{1,2,\ldots,j\}, p_i(t) \neq \text{finished}\}$. Write $c-prop_j(t) := c-prop_j(\sigma_{\leq j}, t)$ if $\sigma_{\leq j}(t)$ fulfills $c-prop_j$ at time t and $\neg c-prop_j(t)$ otherwise. In addition, we assume for $\sigma_i, \sigma_j, \sigma_k \in L_t$ that $\neg c-prop_k(t)$ for all $k > j$ whenever $\neg c-prop_j(t)$ and that $c-prop_i(t)$ for all $i < j$ whenever $c-prop_j(t)$. Collective property lookahead is defined to have access at time t to a largest possible subsequence $\sigma_{\leq j}(t)$ of unfinished input elements such that they collectively fulfill $c-prop_j$ at time t. The first time t where $c-prop_i(t)$ is fulfilled is set to be the preponed release time of σ_i. Thus, in the lookahead case, we have

$$\tau'_i := \min\{t \mid c-prop_i(t)\}. \tag{17}$$

The instance revelation rule of the pure online case has to ensure that $\tau'_i \leq \tau_i$. Collective property lookahead is used when input elements collectively influence which part of the input sequence is seen, e.g., due to their combined size or weight. A typical example for a collective property lookahead arises due to limited surveillance capabilities in material flow systems. For instance, the collective property can then be formulated as "The sum of the lengths of the forthcoming conveying units (e.g., boxes of different sizes) on the conveyor belt is no longer than D metres".

Property lookahead: We introduce this type of lookahead as a generalization of all lookahead types that can be described explicitly. Let $prop_i$ be a time variant property of each input element σ_i. Write $prop_i(t) := prop_i(\sigma_i, t)$ if σ_i fulfills $prop_i$ at time t and $\neg prop_i(t)$ otherwise. The first time t where $prop_i(t)$ is fulfilled is set to be the preponed release time of σ_i. Thus, in the lookahead case, we have

$$\tau'_i := \min\{t \mid prop_i(t)\}. \tag{18}$$

The instance revelation rule of the pure online setting has to ensure that $\tau'_i \leq \tau_i$. Property lookahead is used whenever there is some device that allows us to recognize input elements which fulfill the property. A typical example for a property lookahead in vehicle dispatching arises due to limited information about vehicle arrivals. For instance, the property can then be formulated as "The vehicle is currently located within a distance of at most D kilometres from the dispatching control station".

Observe that in all lookahead types except for time and property lookahead, the release time of an input element σ_i depends on the processing or property status of at least one input element that had been released before σ_i. Thus, the input data record of σ_i is generated on-the-fly during algorithm processing. In fact, it is a key characteristic of online optimization that not all data is known in advance but rather generated and made accessible over time.

3.2.3. Processing mode and order

According to Section 2.1, input element processing is subject to the rules specified in (processing) rule sets P and P' in the absence and presence of lookahead, respectively. Unfortunately, in the literature it is never specified how P and P' exactly look like but tacitly assumed according to the context. For example, let us consider online scheduling with lookahead. While the problem settings in [18,44,46] feature job arrival dates and lead to the possibility of having a processing order different than the arrival order, the problem settings in [45,57] exhibit some kind of job list and the jobs will finally also be processed according to the list order. Clearly, it would be advantageous for the understanding of the problem setting if these rules could be specified explicitly in advance. We provide a verbal classification of the essential difference between P and P'. By definition, the decisions of an algorithm on processing start and end times have to obey

$$\underline{T}_i \leq t_i^{a,start} \leq t_i^{a,finish} \leq \overline{T}_i \tag{19}$$

in the pure online case; in the lookahead case, $\underline{T}_i$ and $\overline{T}_i$ have to be replaced by $\underline{T}'_i$ and $\overline{T}'_i$, respectively. Under lookahead, we have to deal with the questions whether we have to adhere to the order in

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

the input element sequence also in processing (processing order) and whether more than one input element can be processed at a time (processing mode). We explain combinations of the four possible processing modes of single processing, parallel processing, limited parallel processing, and property processing with the two possible processing orders of in-order processing and random-order processing.

Single in-order processing: In single in-order processing, we have to obey the order of input element releases also when processing the input elements and we have to process them one after another. For decisions on σ_i it has to hold that

$$t_i^{a,finish} \leq t_{i+1}^{a,start}. \tag{20}$$

Single random-order processing: In single random-order processing, we are allowed to choose any available unfinished input element for being processed next at any time, but have to respect that only one input element can be processed at a time. For decisions on σ_i and σ_j with $i \neq j$ it has to hold that

$$[t_i^{a,start}, t_i^{a,finish}) \cap [t_j^{a,start}, t_j^{a,finish}) = \emptyset. \tag{21}$$

Parallel in-order processing: In parallel in-order processing, we have to obey the order of input element releases also when processing the input elements, but we may process more than one input element at a time. For decisions on σ_i it has to hold that

$$t_i^{a,start} \leq t_{i+1}^{a,start}. \tag{22}$$

Parallel random-order processing: In parallel random-order processing, we are allowed to choose any available unfinished input element for being processed next at any time and we may process more than one input element at a time. There are no temporal restrictions on processing times which have to hold for decisions on the input elements.

Limited parallel in-order processing: Limited parallel in-order processing is similar to parallel in-order processing, but additionally imposes that at most m input elements can be processed at a time. For decisions on input element σ_i it has to hold that

$$t_i^{a,start} \leq t_{i+1}^{a,start} \tag{23}$$

and additionally for $t \geq 0$ that

$$|\{\sigma_j | t \in [t_j^{a,start}, t_j^{a,finish})\}| \leq m. \tag{24}$$

Limited parallel random-order processing: Limited parallel random-order processing is similar to parallel random-order processing, but additionally imposes that at most m input elements can be processed at a time, i.e., for decisions it has to hold for $t \geq 0$ that

$$|\{\sigma_j | t \in [t_j^{a,start}, t_j^{a,finish})\}| \leq m. \tag{25}$$

Property processing: In property processing, input elements eligible to be processed at time t are marked. Let $proc_i$ be a time-dependent property of each input element σ_i; write $proc_i(t) := proc_i(\sigma_i, t)$ if σ_i fulfills $proc_i$ at time t and $\neg proc_i(t)$ otherwise. Processing start times are coordinated such that $proc_i(t)$ is fulfilled when $t_i^{a,start} = t$ is chosen. For decisions on σ_i it has to hold that

$$t_i^{a,start} \in \{t \geq 0 | proc_i(t)\}. \tag{26}$$

For instance, a priority list for input element processing may be implemented by using the property "The input element has the highest priority among all input elements in the lookahead set" which is based on the specification of a value for an input element's priority in the input element information. Also all previous processing modes and orders can be emulated by property processing using an adequate specification of $proc_i$.

3.2.4. Processing accessibility

We address the question of when input elements are ready to be processed once they have been disclosed. Exact specifications of instance revelation rules r and r' as well as of rule sets P and P' according to Section 2.1 would resolve this issue. However, these specifications are rarely given explicitly in publications. To this end, consider online routing problems. In the problems described in [4,6,37], we find that a decision maker has to wait until the regular release date of a request first before it may be served, whereas in the problem variants given in [52], we learn that one is allowed to serve a request immediately once it is known. We provide a verbal classification of the processing permissions arising by lookahead. The processing accessibility tells us whether there can only be an informational benefit of lookahead, or also an additional processual benefit.

Immediate accessibility: Processing σ_i is possible directly upon receiving it, i.e., we have $\underline{T}_i := \tau_i$ in the pure online case and $\underline{T}'_i := \tau'_i$ in the lookahead case.

Regular accessibility: Processing σ_i is possible only when the regular earliest processing time is reached, i.e., we have $\underline{T}'_i := \underline{T}_i$.

Delayed accessibility: Processing σ_i is not possible directly upon notification, but may be before the regular earliest processing time, i.e., we have $\underline{T}_i > \tau_i$ in the pure online case and $\underline{T}'_i > \tau'_i$ in the lookahead case. An example is to impose a fixed offset t_{off} between information release and earliest possible processing, i.e., $\underline{T}_i = \tau_i + t_{off}$ and $\underline{T}'_i = \tau'_i + t_{off}$.

Although the lower bounds $\underline{T}_i$ and $\underline{T}'_i$ of the processing time intervals from Section 3.2.1 already specify exactly when an input element is available to be processed, we mention the processing accessibility explicitly in order to build a classification scheme (cf. Section 3.4) which will allow us to quickly identify a problem's key characteristics.

3.2.5. Algorithm execution mode

The algorithm execution mode controls at which time instants an algorithm is executed in order to determine the values of the action variables of known input elements. Recall that algorithm execution is concerned with making decisions about what to do with the input elements at which time and not with action execution; action execution runs automatically, but algorithm execution needs initiation. When exact algorithms are applied to instances of $\mathcal{NP}$-hard subproblems, we have to ensure that real-time requirements are not violated.

Cyclic execution: Algorithm execution is carried out cyclically, i.e., at all times

$$t_i^{exec} = i \cdot t^{cycle} \tag{27}$$

for $i = 0, 1, 2, \ldots$ where t^{cycle} is the cycle time. It is possible that an arbitrary number of new input elements accumulates during algorithm executions or that no new input element arrives at all. Execution may still be worthwhile because actions different from NULL may be determined for known input elements which have not yet been assigned an action.

Full buffer execution: Algorithm execution is performed every time that the number of elements in the lookahead set L_t reaches a prescribed limit $c \in \mathbb{N}$, i.e., at all times in

$$\{t^{exec} \geq 0 | |L_{t_{exec}}| = c, |L_{t_{exec}-\epsilon}| < c\} \tag{28}$$

with sufficiently small $\epsilon > 0$. At the end of the input sequence, it has to be assured that none of the input elements σ_i exhibits $a_i(t) = \text{NULL}$.

Discrete event execution: Algorithm execution is triggered by events occurring at discrete times. From a technical point of view, an event detecting device must be installed. Release of a new input element and finished processing of an input element are typical events in basic online optimization. In more complex settings,

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

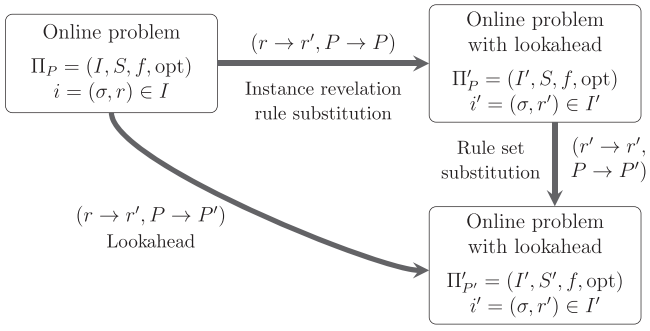

Fig. 2. Change of problem instances and problem through application of lookahead.

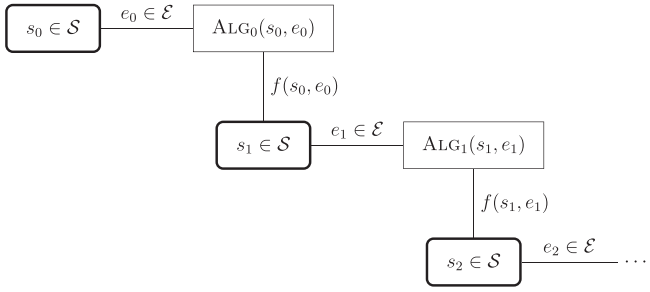

Fig. 3. State trajectory with associated state transition function and algorithm evaluations in the process model for online optimization with lookahead.

additional events (breakdowns, failures) can also be considered as events. Denote by $(t_i^e)_{i \in \mathbb{N}}$ the sequence of time instants at which events are notified, then algorithm execution takes place at times

$$t_i^{exec} = t_i^e + \epsilon \tag{29}$$

for $i \in \mathbb{N}$ with sufficiently small $\epsilon > 0$. Practically all types of algorithm execution modes can be traced back to discrete event execution by appropriate definition of the set of events (Fig. 2).

3.3. Discrete event process model

Instantiations of the basic modeling elements are combined in a process model which describes the interaction of the modeling elements over time as a result of the specific forms of lookahead type, implied processing characteristics and prescribed algorithm execution mode. The system is assumed to operate on an event-driven basis. Starting with initial state $s_0 \in \mathcal{S}$, Fig. 3 schematically illustrates how the state trajectory $(s_0, s_1, s_2, \ldots)$ evolves as a result of events arriving in event sequence $(e_0, e_1, e_2, \ldots)$ and related computations of algorithm ALG: upon arrival of an event $e \in \mathcal{E}$, the system proceeds from its current state $s \in \mathcal{S}$ to successor state $s' \in \mathcal{S}$ by evaluating the state transition function $f(s, e)$. However, computing the successor state at time t as $f(s, e)$ implicitly requires a preceding evaluation of $\text{ALG}_t(s, e) \in \mathcal{A}^{n_t}$ where n_t is the number of known unfinished input elements because the successor state also contains a specification of the action variables for yet unfinished input elements. Upon reaching the successor state, the system awaits the arrival of a new event causing the system to undergo the same sequence of steps again.

The computational effort of an algorithm caused by a state transition typically depends on the type of the encountered event e: the arrival of a new input element suggests to solve a snapshot optimization problem in order to determine the action variable for the new input element and to redetermine action variables for still unfinished input elements; finished processing of an input element will not cause an algorithm to spend excessive computational resources since all remaining actions are expected to remain the same as they have been computed previously on the same informational basis.

Information about the evolution of the objective value which is incurred by processing the elements of the input sequence can be tracked by monitoring the objective state component s^{obj} of state $s = (s^{in}, s^{sys}, s^{obj})$. Typically, when an event corresponds to a finished processing of an input element, a change in the objective value can be observed as a result of the associated state transition, whereas when an event corresponds to a new input element release, no immediate change in the objective value will occur since processing of this input element is still pending.

3.4. Classification scheme

Similar to the classification scheme for scheduling, we provide a classification scheme for the modeling in online optimization with lookahead that takes into account their characteristics with respect to lookahead $(r \to r', P \to P')$. We propose a four-position scheme $\alpha|\beta|\gamma|\delta$ in order to quickly indicate the qualitative characteristics of a problem under lookahead as compared to the (reference) problem without lookahead:

Lookahead type α: *req* for request lookahead, *req-b* for request lookahead in batches, *time* for time lookahead, *col* for collective property lookahead, *prop* for property lookahead.

Processing mode and order β: *sngl/ord* for single in-order processing, *sngl/rnd* for single random-order processing, *prl/ord* for parallel in-order processing, *prl/rnd* for parallel random-order processing, *prl-ltd/ord* for parallel limited in-order processing, *prl-ltd/rnd* for parallel limited random-order processing, *pp* for property processing.

Processing accessibility γ: *im* for immediate accessibility, *reg* for regular accessibility, *dly* for delayed accessibility.

Algorithm execution mode δ: *cyc* for cyclic algorithm execution, *full* for full buffer algorithm execution, *discr* for discrete event algorithm execution.

With respect to the definition of optimization problems and lookahead in Section 2.1, α indicates the instance revelation rule substitution from r to r'; β and γ are established by rule set P' which constitutes the conditions for input element processing such that compliance with the feasible set under lookahead is ensured; δ is part of the solution routine and not of the problem. In the literature, lookahead is mainly defined with respect to the topic of a paper and an explicit specification of processing characteristics is tacitly assumed.

Table 2 classifies lookahead concepts as found in papers on online optimization with lookahead. In accordance with the deterministic perception of lookahead prevalent in online optimization (cf. Section 1.1), it was possible to cover all notions of lookahead concepts that were found in papers on online optimization problems that have been extended by lookahead. Intentionally, the scheme does not capture information about future events which is given in a non-deterministic, probabilistic or scenario-based form (e.g., in stochastic programming, dynamic programming, model predictive control, or interval analysis).

4. Performance measurement in online optimization

We subsequently discuss how algorithms should be evaluated in order to comply with needs of practitioners which have to select the most suitable algorithm for their application and have to specify the amount of lookahead that should be facilitated technologically.

The academic standard for the performance of online algorithms is competitive analysis [13] where algorithm ALG is put into

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

Table 2
Classification of concepts in papers on online optimization with lookahead.

Reference	α	β	γ	δ
Routing and transportation				
Allulli et al. [4,5]	time	sngl/rnd	reg	discr
Ausiello et al. [6]	time	sngl/rnd	reg	discr
Jaillet and Lu [36] (advanced information)	prop	sngl/rnd	reg	discr
Jaillet and Wagner [37] (advanced information)	prop	sngl/rnd	reg	discr
Tinkl [52] (lookahead (LA) by order)	req	sngl/rnd	im	cyc
Scheduling				
Coleman [18]	req	sngl/rnd	reg	discr
Li et al.[44]	time	prl/rnd	reg	discr
Mandelbaum and Shabtay [45] (adaptive LA)	req	prl-ltd/ord	reg	cyc
Mandelbaum and Shabtay [45] (non-adaptive LA)	req-b	prl-ltd/ord	reg	cyc
Mao and Kincaid [46]	req	sngl/rnd	reg	discr
Motwani et al. [47] (finite LA)	req	sngl/ord	im	cyc
Yang et al. [54] (head-of-the-line)	req	sngl/rnd	im	cyc
Zheng et al. [57]	time	sngl/ord	reg	discr
Zheng et al. [58]	time	sngl/rnd	reg	discr
Data structures				
Albers [2,3] (weak LA)	req	sngl/ord	reg	cyc
Albers [3] (strong LA)	col	sngl/ord	reg	cyc
Breslauer [14] (natural LA)	col	sngl/ord	reg	cyc
Kiniwa et al. [41]	req	sngl/rnd	im	cyc
Torng [53]	req	sngl/ord	reg	cyc
Yeh et al. [55]	req-b	sngl/rnd	im	cyc
Young [56] (resource-bounded LA)	col	sngl/ord	reg	cyc
Data transfer				
Dooly et al. [20] (oracle)	req	sngl/ord	reg	cyc
Imrek and Nemeth´ [34]	time	sngl/ord	reg	discr
Packing				
Grove [29]	col	sngl/rnd	im	cyc
Gutin et al. [31]	req-b	sngl/rnd	im	cyc
Lot sizing				
Ahlroth et al. [1]	time	prl/ord	im	cyc
Metrical task systems				
Ben-David and Borodin [9]	req	prl-ltd/ord	reg	cyc
Koutsoupias and Papadimitriou [42]	req	sngl/ord	reg	cyc
Graph theory				
Chung et al. [17] (window index)	col	sngl/ord	reg	cyc
Halldorssón and Szegedy [32]	req	sngl/ord	reg	cyc
Halldorssón and Szegedy [32] (buffer)	req-b	sngl/ord	reg	cyc
Irani [35]	req	sngl/ord	reg	cyc

competition with an optimal offline algorithm OPT which knows σ in advance. ALG is called c-competitive if there is an $a \in \mathbb{R}$ such that

$$\mathrm{ALG}[\sigma] \le c \cdot \mathrm{OPT}[\sigma] + a \tag{30}$$

for all input sequences σ. The competitive ratio c_r of ALG is the greatest lower bound over all c such that ALG is c-competitive. There are many disadvantages of the competitive ratio [21,24] with respect to practical purposes: results are overly pessimistic because pathologic instances are decisive, competing with OPT is practically irrelevant, overall algorithm behavior is neglected, and competitive analysis is impossible in practical applications. To overcome these shortcomings, we introduce a two-sided performance measurement approach which summarizes global algorithm behavior (over all instances) and also considers local quality (instancewise comparison). The two-sided concept is discussed in detail in [22].

Definition 8 (*Objective value/performance ratio*). Let $\Pi = (I, S, f, \mathrm{opt})$ be an optimization problem, let $i \in I$ be an instance of Π, and let ALG_1, ALG_2 be two algorithms for Π choosing solutions $s_{\mathrm{ALG}_1}(i)$, $s_{\mathrm{ALG}_2}(i) \in S(i)$ on i, respectively.

(a) $v_{\mathrm{ALG}_1}(i) := f(i, s_{\mathrm{ALG}_1}(i))$ is called objective value of ALG_1 with respect to i.

(b) $r_{\mathrm{ALG}_1,\mathrm{ALG}_2}(i) := \frac{f(i,s_{\mathrm{ALG}_1}(i))}{f(i,s_{\mathrm{ALG}_2}(i))}$ is called performance ratio of ALG_1 relative to ALG_2 with respect to i.

Traditionally, there are no probabilities for instances in online optimization. Hence, the best way of dealing with the absence of any probabilistic information is to assume the maximum entropy distribution over the set of all possible instances because it minimizes the amount of a-priori knowledge contained in the distribution [38,39]. From the definition of the entropy and the constraint that all probabilities add up to 1, we conclude by Lagrangian relaxation that the uniform distribution over I is the maximum entropy distribution among all distributions with support I. For finite I, we obtain counting results saying how many of all instances yield a certain objective value or performance ratio [33]. The counting distribution function[1] displays these frequency information over I:

Definition 9 (*Counting distribution functions*). Let $\Pi = (I, S, f, \mathrm{opt})$ be an optimization problem, let $i \in I$ be an instance of Π, let ALG_1, ALG_2 be two algorithms for Π choosing solutions $s_{\mathrm{ALG}_1}(i), s_{\mathrm{ALG}_2}(i) \in S(i)$ on i, respectively, and let I be a discrete set.

(a) The counting distribution function of the objective value of ALG_1 over I is given by $F_{\mathrm{ALG}_1} : \mathbb{R} \to [0,1]$ with

$$F_{\mathrm{ALG}_1}(v) := \frac{\sum_{i \in I} \mathbf{1}_{(-\infty, v]}(v_{\mathrm{ALG}_1}(i))}{|I|}. \tag{31}$$

(b) The counting distribution function of the performance ratio of ALG_1 relative to ALG_2 over I is given by $F_{\mathrm{ALG}_1,\mathrm{ALG}_2} : \mathbb{R} \to [0,1]$ with

$$F_{\mathrm{ALG}_1,\mathrm{ALG}_2}(r) := \frac{\sum_{i \in I} \mathbf{1}_{(-\infty, r]}(r_{\mathrm{ALG}_1,\mathrm{ALG}_2}(i))}{|I|}. \tag{32}$$

Since we are interested in comparing algorithms with lookahead relative to algorithms without lookahead, we can select in Definition 9 for ALG_1 some algorithm with lookahead and for ALG_2 some algorithm without lookahead.

Fig. 4 shows that two algorithms are compared by examining the relative positions of their counting distribution function plots to each other in the case of the objective value (left side of Fig. 4), and by partitioning the set of all instances into those which favor $\mathrm{ALG}_1 (r < 1)$, those which favor $\mathrm{ALG}_2 (r > 1)$ and those which are indifferent $(r = 1)$ in the case of the performance ratio (right side of Fig. 4).

The counting distribution of the performance ratio implicitly includes competitive analysis because the largest occurring performance ratio coincides with the competitive ratio. In this sense, analyzing the counting distribution function of the performance ratio is an extension of competitive analysis. Hence, whenever worst-case system performance is critical, it is recommendable to have a detailed look on the worst performance ratios in order to hedge against worst-case scenarios as intended by competitive analysis. We conclude that counting distribution functions shall be considered as a supplementary performance measurement method giving additional information about ranges, frequencies and variability, and not as an exclusive alternative to competitive analysis.

In particular, we now have an approach which summarizes the global behavior of an algorithm over all instances, but also does not lose sight of local quality with respect to particular instances. This is especially worthwhile in analyzing control strategies for

[1] The indicator function $\mathbf{1}_A(x)$ is 1 if $x \in A$ and 0 otherwise.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

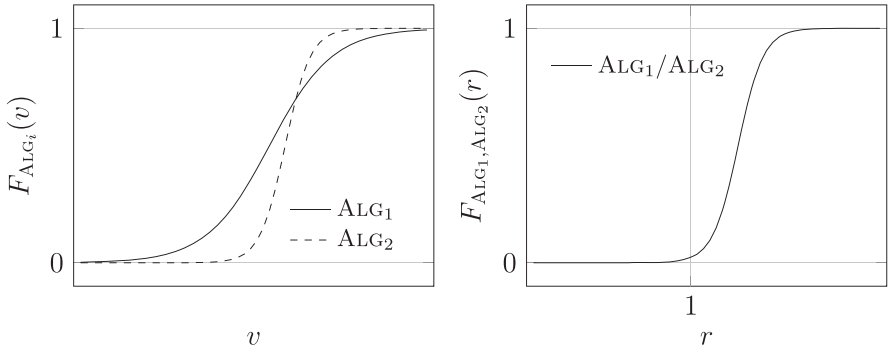

Fig. 4. Counting distribution functions of objective value of ALG$_1$ and ALG$_2$ (left) and of performance ratio of ALG$_1$ relative to ALG$_2$ (right).

operative applications where one is interested in the behavior of daily operations rather than in worst-case scenarios. By computing statistical key figures (such as confidence intervals, variance, quantiles, cf. Online Appendix A) after conducting numerical experiments, we can also obtain valuable information about system behavior in general. Additionally, competitive analysis is often no longer practicable once the problem settings get slightly more complicated. But definitely we also need to have a suitable performance measurement method in these cases.

Furthermore, the approach has the advantage that algorithms with arbitrary lookahead levels can be compared directly such that there is no dependence on the performance of an optimal offline algorithm OPT. This fact makes the approach considerably versatile with respect to practical questions arising through the installation of lookahead devices. Hence, we often find ourselves in settings with small lookahead where problem sizes can be bounded by practical considerations. In fact, in applications we are often much more interested in the question "What can be gained through additional lookahead?" rather than in the question "What is lost through not having the offline situation?" because we are far away from the offline situation anyway. Nonetheless, the decision maker is free to select the information regimes of ALG_1 and ALG_2 as required by the research question. This also means that in the case of full information, one has to deal with the burden of computational complexity when performance shall be analyzed with respect to exact algorithms; for heuristic algorithms as a benchmark this issue is not a problem.

5. Computational results and information pool

Based on the taxonomy for online optimization with lookahead, the general modeling framework and the performance measurement approach, we present the results of a series of numerical experiments on standard online optimization problems. In detail, we will now report on the results for the bin packing and the traveling salesman problem in Sections 5.1 and 5.2. Two types of analysis are applied in order to ensure a comprehensive view on algorithm behavior under different information regimes: average results portray the overall behavior of algorithms under different amounts of lookahead; distributional results submit a more precise picture by including frequency information on observed objective values and performance ratios. A statistical summary of the computational results is also given in the Online Appendix A. Results for the ski rental, paging and scheduling problem cannot be discussed due to restricted space. For further details and additional analysis in all problems, see the discussion in [22]. However, in Section 5.3, we will summarize the overall findings of our computational experiments in an information pool which may help in future applications to preestimate the potential of lookahead.

Computational experiments were performed on a computer with AMD Phenom II X6 1100 T 3.31 GHz processor and 16 GB RAM under Microsoft Windows 7 (64-bit). Algorithms were implemented in C++; IP and MIP formulations were solved using IBM ILOG CPLEX 12.5 with a time limit of 120 s. All results are collected in an information pool which gives an overview for observable lookahead effects in different applications. Whenever a problem arises in practice, we may look into the information pool in order to preestimate from the involved elementary problems how large a lookahead effect is to be expected.

5.1. Online bin packing with lookahead

A fundamental packing problem is (one-dimensional) bin packing [40] where the task is to pack the items $\sigma_1, \sigma_2, \ldots, \sigma_n$ for $n \in \mathbb{N}$ with sizes $s_i \in (0,1]$ for $i = 1, \ldots n$ into a minimum number of unit-capacitated bins. In the numerical analysis, we consider two rule sets: item permutations may be allowed when items are physically small such that sorting is possible, or item permutations may be forbidden when items are too large to be rearranged.

For $n \in \mathbb{N}$, the set of all input sequences of length n is given by $\Sigma_n = \{(\sigma_1, \sigma_2, \ldots, \sigma_n) | \sigma_i := s_i \in (0,1], i = 1, \ldots, n\}$ and comprises all item sequences of length n where σ_i is identified with the size of the ith item. In the online version, only σ_i with $i = 1, \ldots, n$ is known when σ_i has to be packed. When σ_i with $i = 1, \ldots, n$ has to be packed in the online version with lookahead of size $l \in \mathbb{N}$ under order preservation, also $l-1$ successive items $\sigma_{i+1}, \ldots, \sigma_{i+l-1}$ are known if $i+l-1 \le n$, otherwise σ_i and additional $n-i$ successive items $\sigma_{i+1}, \ldots, \sigma_n$ are known. When known items may be packed in arbitrary order, then at the ith packing time, the l unpacked items from $\sigma_1, \sigma_2, \ldots, \sigma_{i+l-1}$ with $i = 1, \ldots, n$ are known if $i+l-1 \le n$, otherwise the $n-i+1$ unpacked items from $\sigma_1, \sigma_2, \ldots, \sigma_n$ are known. In the offline version, all items are known at the beginning. The instance revelation rule r in the online case and r' in the lookahead case is

r:=Initially, σ_1 is known; a new item is revealed when a known one is packed;
r':=Initially, $\sigma_1, \ldots, \sigma_l$ are known; a new item is revealed when a known one is packed.

The rule set P in the online case is trivial, in the lookahead case we have to distinguish between allowed permutations (P'_1) and forbidden permutations (P'_2), i.e.,

P:={Pack the known item};
P'_1:={Pack one of the known items};
P'_2:={Pack the known item which has been released earliest}.

According to the modeling framework, the lookahead setting is *req* | *sngl/rnd* | *im* | *discr* (or equivalently *req* | *sngl/rnd* | *im* | *cyc*). The input information of an input element is given by s_i. In the online case, we have $\tau_i = i$ and $T_i = i+\epsilon$; in the lookahead case, we have

$\tau_i = \max\{1, i-l\}$ and $\underline{T}'_i = \max\{1, i-l\}+\epsilon$, $\overline{T}'_i = \infty$ with sufficiently small $\epsilon > 0$. In the event set, we take into account the events of a new item arrival and finished assignment of an item; both event types coincide as long as there are still unreleased requests. The action space corresponds to $\{1, 2, \ldots, n\}$, and an action amounts to choosing the index of the bin into which the next item should be put. A state holds information on the current lookahead set and configuration of previously opened bins. For a formal representation, see [22].

A decision by an algorithm is required for every item that has to be packed and consists of selecting the bin in which this item should be put. Because a bin once opened is never closed again, each of the following algorithms can be used irrespective of whether item permutations are allowed or not: in case of forbidden permutations the items must be packed in their order of release, but it is still possible to employ the algorithms which determine an item to bin assignment based on a packing order differing from the release order. This is due to the fact that we can first fictively reserve the space in the bins for the items as prescribed by the assignment, but then in the packing process we just adhere to the item release order.

Before we specify the bin packing algorithms operating under lookahead, we first recall the following two classical bin packing algorithms for the pure online case without lookahead:

FirstFit (Ff): If there is at least one open bin that can accommodate the item to be packed, put the item in the bin that was opened first among these bins; otherwise open a new bin and put the item in the new bin [19].

BestFit (BF): If there is at least one open bin that can accommodate the item to be packed, put the item in the fullest among these bins; otherwise open a new bin and put the item in the new bin [19].

These two algorithms can now easily be generalized to their lookahead versions. For $l=1$ we obtain the pure online versions, for $l=n$ the algorithms coincide with corresponding offline algorithms.

FirstFit$_l$(Ff$_l$): Sort the items in the lookahead by non-increasing size and fictively pack them with Ff. If the item to be packed is put in a new bin, open a new bin and put the item in the new bin; otherwise put the item in the bin from the fictive assignment [19].

BestFit$_l$(BF$_l$): Sort the items in the lookahead by non-increasing size and fictively pack them with BF. If the item to be packed is put in a new bin, open a new bin and put the item in the new bin; otherwise put the item in the bin from the fictive assignment [19].

Optimal$_l$(Opt$_l$): In Fig. 5, specify N according to the number of seen items, N^o according to the number of already used bins, s according to the sizes of the seen items, and f according to the current fill levels of the already used bins. Solve the resulting IP formulation in Fig. 6. If the item to be packed is put in a new bin, open a new bin and put the item in the new bin; otherwise put the item in the bin from the obtained assignment.

Optimal$'_l$(Opt$'_l$): In Fig. 5, specify N according to the number of seen items, N^o according to the number of already used bins, s according to the sizes of the seen items, and f according to the current fill levels of the already used bins. Solve the IP formulation in Fig. 6 after extending it with expressions from Fig. 7 as follows: Replace Objective Function (33) by (38) and include Constraints (39)–(43). If the item to be packed is put in a new bin, open a new

Sets and parameters

I	set of items with $I = \{1, \ldots, N\}$
J	set of potential new bins with $J = \{1, \ldots, N\}$
J^o	set of already used bins with $J^o = \{1, \ldots, N^o\}$
L	set of fill levels ℓ with $L = \{0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$
N	number of items (N is an upper bound for the number of potential new bins)
N^o	number of bins already used
s_i	size of item $i \in I$
f_{j^o}	fill level of already used bin $j^o \in J^o$
w_ℓ	weight of fill level $\ell \in L$ with $w_{0.7} = 0.01$, $w_{0.75} = 0.02$, $w_{0.8} = 0.04$, $w_{0.85} = 0.08$, $w_{0.9} = 0.16$, $w_{0.95} = 0.32$

Variables

$$x_{ij}/x^o_{ij^o} = \begin{cases} 1 & \text{if item } i \in I \text{ is put in bin } j \in J/ \\ & \text{already used bin } j^o \in J^o, \\ 0 & \text{else} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if bin } j \in J \text{ is used,} \\ 0 & \text{else} \end{cases}$$

$$\alpha_{j\ell}/\alpha^o_{j^o\ell} = \begin{cases} 1 & \text{if bin } j \in J/ \text{ already used bin } j^o \in J^o \text{ has fill level in} \\ & [\ell, \ell+0.05) \text{ for } \ell \in L \text{ with } \ell \neq 0.95, \\ & \text{or } [\ell, \ell+0.05] \text{ for } \ell = 0.95, \\ 0 & \text{else} \end{cases}$$

Fig. 5. Sets, parameters and variables in the IP formulation of the bin packing problems [23].

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

$$\min \quad \sum_{j\in J} y_j + |J^o| \tag{33}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ij} + \sum_{j^o\in J^o} x^o_{ij^o} = 1 \qquad i \in I \tag{34}$$

$$\sum_{i\in I} s_i x_{ij} \le y_j \qquad j \in J \tag{35}$$

$$\sum_{i\in I} s_i x^o_{ij^o} \le 1 - f_{j^o} \qquad j^o \in J^o \tag{36}$$

$$x_{ij}, x^o_{ij^o}, y_j \in \{0,1\} \qquad i \in I,\ j \in J,\ j^o \in J^o \tag{37}$$

Fig. 6. IP formulation of the bin packing problem.

$$\min \quad \sum_{j\in J} y_j + |J^o| - \tfrac{1}{|J|+|J^o|}\Big(\sum_{j\in J}\sum_{\ell\in L} w_\ell \alpha_{j\ell} + \sum_{j^o\in J^o}\sum_{\ell\in L} w_\ell \alpha^o_{j^o\ell}\Big) \tag{38}$$

$$\sum_{\ell\in L} \alpha_{j\ell} \le 1 \qquad j \in J \tag{39}$$

$$\sum_{\ell\in L} \alpha^o_{j^o\ell} \le 1 \qquad j^o \in J^o \tag{40}$$

$$\ell\alpha_{j\ell} \le \sum_{i\in I} s_i x_{ij} \qquad j \in J,\ \ell \in L \tag{41}$$

$$\ell\alpha^o_{j^o\ell} \le \sum_{i\in I} s_i x^o_{ij^o} \qquad j^o \in J^o,\ \ell \in L \tag{42}$$

$$\alpha_{j\ell}, \alpha^o_{j^o\ell} \in \{0,1\} \qquad j \in J,\ j^o \in J^o,\ \ell \in L \tag{43}$$

Fig. 7. Objective function and additional constraints for the modified IP formulation of the bin packing problem (see also [23]).

bin and put the item in the new bin; otherwise put the item in the bin from the obtained assignment.

While Opt_l only considers the number of bins in the objective, Opt'_l secondarily searches for an item to bin assignment with bins as full or empty as possible, but not with medium fill levels. Thereby, the initial position for successive steps is improved because in case of large item sizes, empty bins are more valuable. Using a surrogate objective function in order to improve the initial position for future steps is due to Esen [23].

In addition to these algorithms, we also consider their batched versions in order to check whether continuous information release is necessary or whether information release in blocks (so-called batches) of items suffices to obtain satisfactory results. If algorithms operate under batched lookahead, we indicate them with an added suffix B in the algorithm name, e.g., $\text{BF}_{10,B}$ means that the BestFit$_{10}$ algorithm is applied once for each batch of 10 items, and only after all of the 10 items have been put into a bin a new batch of 10 items is released.

We now present the computational results for the algorithm families Ff_l, BF_l, Opt_l, Opt'_l and their respective batched versions under variable size l of the lookahead set. We select the settings of $n=25$ and $n=100$ items per sequence; each setting features $m=1000$ independently drawn item sequences. We discuss results for $n=25$ and refer to [22] for $n=100$. For $n=25$, all lookahead sizes $l \in \{1,5,10,15,20,25\}$ are tested in order to quantify the value of additional lookahead.

Average results: Average algorithm behavior under variable lookahead size l is displayed on the left side of Fig. 8 for the case of allowed item permutations. The effect of lookahead is considered nonsignificant irrespective of the selected algorithm family which can be seen by the negligible dimension of the reduction in

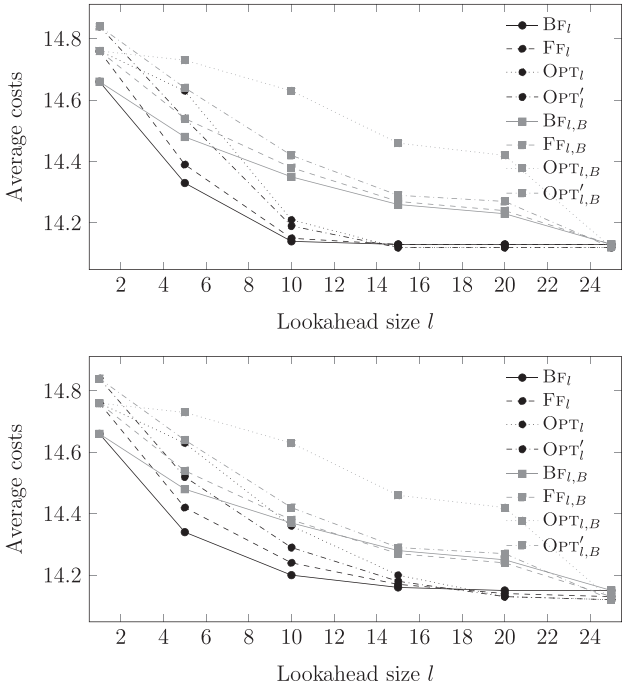

Fig. 8. Average costs for different lookahead sizes and $n=25$ in the classical bin packing problem when item permutations are allowed (top) and when item permutations are forbidden (bottom).

average costs for increasing lookahead size l. This effect can be explained by the fact that a large number of conditions concerning the combination of item sizes have to be met in order to lead to a

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

saving of one bin. In typical item sequences such conditions are unlikely to occur. Between the pure online and offline situations only 0.53 (BF_1 vs. BF_{25}) to 0.72 (OPT'_1 vs. OPT'_{25}) bins could be saved on average which is an improvement of 3.6–4.7%, respectively. Average algorithm behavior in the case of forbidden item permutations on the right side of Fig. 8 is nearly identical; e.g., OPT_{25} needed an average of 14.12 bins in case of allowed item permutations and now it needs 14.2 bins in case of forbidden item permutations which is a relative deterioration of less than 1%. Recall also that in Example 1 there was no value of permuting items at all for the case of two item sizes. Hence, we will restrict ourselves to the case of allowed item permutations from now on.

We also find that simple algorithms like BF_l and FF_l outperform the exact reoptimization methods OPT_l and OPT'_l for lookahead sizes $l<15$. BF_l produces the best overall results in this information regime. OPT_l and OPT'_l beat the rule-based algorithms only for $l\geq 15$ by a slight margin. Hence, we infer that exact reoptimization is not needed when too many future items are unseen. The stability granted by BF_l and FF_l for small to medium lookahead sizes which is achieved by packing large items first proves advantageous over the "local" optimality of exact solutions to snapshot problems; these solutions are fragile once the situation changes due to newly announced lookahead items. Optimality of partial solutions becomes important only if their benefit cannot be undone by future decisions on upcoming items. Algorithms collectively show a decreasing marginal benefit from lookahead, i.e., the first lookahead units are most valuable and sufficient to drive item to bin assignments towards an optimal solution.

To take full advantage of lookahead, it is recommended to use regular lookahead rather than the batched type: for instance, BF_{10} shows an improvement of 3.5% compared to the online case, while its batched version counterpart $BF_{10,B}$ only accounts for a 2.1% improvement. Exclusive benefit from lookahead cannot be guaranteed. However, instances with a deterioration are encountered rarely.

A summary of statistical key figures for the number of used bins can be found in Table A6 (allowed item permutations) and Table A7 (forbidden item permutations) of Online Appendix A.

Distributional results: The distributional results with respect to the objective value and the performance ratio are shown exemplarily for algorithm classes BF_l and OPT_l in Fig. 9. Since FF_l behaves similar to BF_l and OPT_l behaves similar to OPT'_l, we omit their figures. The plots are affirmative to the minor positive effect of lookahead. On the left side of Fig. 9, this can be seen from the relative closeness of the plots of two successive lookahead levels to each other. This tells us that with increasing lookahead size l, only a small proportion of all item sequences leads to a bin saving. Thus, only a slight left shift of the distribution functions is observed for increasing l. For medium to large lookahead, differences in the counting distribution functions of the costs are even hardly perceivable. For instance, the counting distribution functions of BF_{10}, BF_{15}, BF_{20} and BF_{25} all seem to coincide in their visual representations.

On the right side of Fig. 9, we compare the algorithms with lookahead relative to their pure online versions ($l=1$). We observe that the largest part of the item sequences lead to performance ratios within $[0.9, 1]$, irrespective of the lookahead level. Hence, there is no great potential for improving the outcome of online algorithms by additional lookahead to more than 10% on single instances with respect to the performance ratio. However, we find that there are at least some few item sequences where lookahead can yield an improvement of up to 10% which was not yet clear from the average results. The potential value of the first lookahead units especially becomes obvious when we consider the curves for lookahead size $l=5$ on the right side of the figure. For BF_5/BF_1, around 35% of the input sequences lead to a performance ratio smaller than 1; for OPT_5/OPT_1, this percentage is around 20%. However, from the fact that the curve for OPT_5/OPT_1 in the lower right diagram of Fig. 9 has some mass on performance ratios larger than 1, we also conclude that there are some few item sequences where additional lookahead may even cause a (slight) deterioration in the number of bins used.

In total, the satisfactory behavior of the online algorithms can be explained by two reasons: first, since any bin is left open forever, a bin utilization of each bin close to 100% (except for the last bin if it was opened towards the end of the packing process and if the remaining items were not large enough to fill that last bin) is probable in the long run also in the online setting and the implications of bad decisions are either unnoticeable or rather small. Second, it is known that already the competitive ratio of BF and FF is $\frac{17}{10}$ and that of BF_n and FF_n is $\frac{11}{9}$ [40,50]. Concluding, we recall that throughout all experiments except for the pure offline case

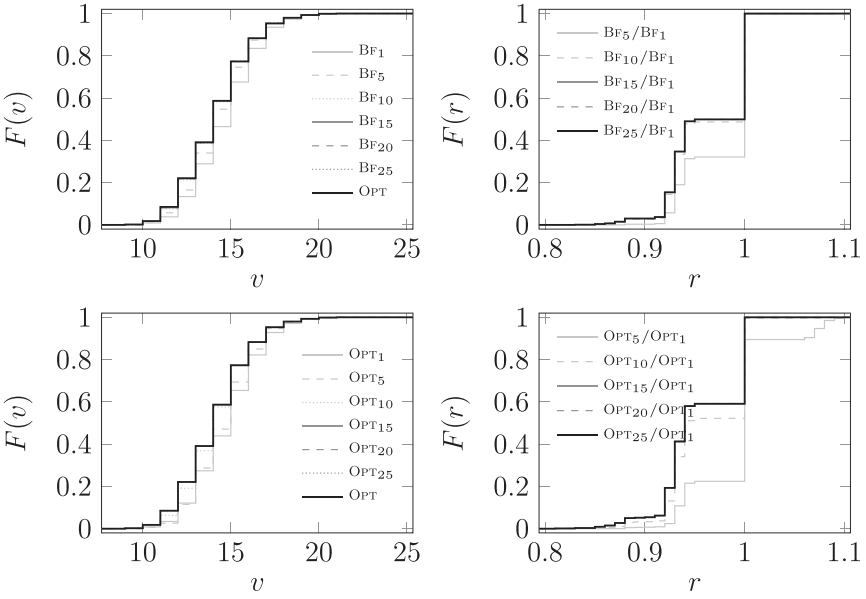

Fig. 9. Empirical counting distribution functions of costs (left) and performance ratios relative to the online case (right) for $n=25$ in bin packing when item permutations are allowed.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

there was no surplus at all of using exact reoptimization methods (Opt_l, Opt'_l) rather than the rule-based heuristics (Ff_l, BF_l). Thus, we recommend to use BF_l in the first place for lookahead sizes $l \leq 15$. For $l > 15$, BF_l also produces high quality packings which are beaten only very slightly (by less than 0.1%) by Opt_l and Opt'_l for lookahead sizes that bring the problem close to the offline situation.

A summary of statistical key figures for the performance ratios relative to the online case can be found in Table A4 (allowed item permutations) and Table A5 (forbidden item permutations) of Online Appendix A.

5.2. Online traveling salesman problem with lookahead

The traveling salesman problem (TSP) seeks to find a round trip (also called tour) for a given set of locations to be visited (also called requests) such that some cost depending on the total travel distance is minimized [43]. For $\sigma = (\sigma_1, \sigma_2, \ldots)$, let a request σ_i with $i \in \mathbb{N}$ correspond to a point x_i in a space $\mathcal{M}$ with metric $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$. The TSP then consists of visiting the points of all requests with a server in a tour of minimum length starting and ending in some distinguished origin $o \in \mathcal{M}$.

For $n \in \mathbb{N}$, the set of all input sequences of length n is given by $\Sigma_n = \{(\sigma_1, \sigma_2, \ldots, \sigma_n) | \sigma_i := x_i \in \mathcal{M}, i = 1, \ldots, n\}$ and comprises all request sequences of length n where σ_i is identified with point $x_i \in \mathcal{M}$ to be visited. In the online version, only σ_i with $i = 1, \ldots, n$ is known when σ_i has to be visited. In the online version with lookahead of size l, when for the ith time it has to be decided which location to be visited, the l unvisited locations from $\sigma_1, \sigma_2, \ldots, \sigma_{i+l-1}$ with $i = 1, \ldots, n$ are known if $i + l - 1 \leq n$, otherwise the $n - i + 1$ unvisited locations from $\sigma_1, \sigma_2, \ldots, \sigma_n$ are known. In the offline version, all locations are known at the beginning. The instance revelation rule r in the online case and r' in the lookahead case is
r:=Initially, σ_1 is known ; a new request is revealed when the known one is visited;
r':=Initially, $\sigma_1, \ldots, \sigma_l$ are known; a new request is revealed when a known one is visited.

The rule set P in the online case is trivial and does not give any degrees of freedom to the decision maker, in the lookahead case (P') we consider a rule set substitution, i.e.,

P:={Visit the known request};
P':={Visit one of the known requests}.

According to the modeling framework, the lookahead setting is *req | sngl/rnd | im | discr* (or equivalently *req | sngl/rnd | im | cyc*). The input information of an input element is given by x_i. In the online case, we have $\tau_i = i$ and $T_i = i + \epsilon$; in the lookahead case, we have $\tau_i = \max\{1, i - l\}$ and $\underline{T}'_i = \max\{1, i - l\} + \epsilon$, $\overline{T}'_i = \infty$ with sufficiently small $\epsilon > 0$. Hence, the server is assumed to travel at infinite speed. In the event set, we take into account the events of a new request arrival and finished service of a request; both event types coincide as long as there are still unreleased requests. The action space corresponds to $\{1, 2, \ldots, l\}$, and an action amounts to choosing the index of the request to be visited next in an indexed representation of the lookahead set. A state holds information on the current lookahead set, server position, and the distance traveled thus far. For a formal representation, see [22].

A decision by an algorithm is required when the server starts initially or a request is reached; it consists of selecting the request to be visited next from the set of known requests. Any online algorithm without lookahead is trivial since it only sees the current request and has to visit it. In this sense, all algorithms collapse into a first come first served strategy for $l = 1$; for $l = n$ offline algorithms are obtained.

NearestNeighbor$_l$(Nn_l): From the requests in the lookahead, choose a request closest to the server's current location next [43].

Insertion$_l$(Ins_l): Construct a Hamiltonian path H visiting each of the requests in the lookahead starting in the current server location and ending in o as follows: at the beginning, H consists of the invariant starting and ending point only. Insert a request whose distance to the starting point is largest possible into H. Successively insert remaining requests by choosing in each iteration a request whose smallest distance to a request in H is largest and insert it at a best possible position in terms of a smallest tour length increase. Finally, choose the first request from H following the starting point next [43].

2Opt_l: Construct a Hamiltonian path H visiting each of the requests in the lookahead starting in the current server location and ending in o as follows: obtain H initially by Nn_l or Ins_l. Until no further improvement is possible, choose two requests from H, reverse the order of requests between them to obtain H', and if H' is shorter than H, then set H:=H'. Finally, choose the first request from H following the starting point next [43].

3Opt_l: Construct a Hamiltonian path H visiting each of the requests in the lookahead starting in the current server location and ending in o as follows: obtain H initially by Nn_l or Ins_l. Until no further improvement is possible, choose three edges from H such that neither of them is incident to the starting or ending point and reorganize H by forming three new edges using the requests incident to the three edges, choosing an order for the three new edges and adjusting the request order between the edges such that a feasible Hamiltonian path H' is obtained if possible at all, and if H' is feasible and shorter than H, then set H:=H'. Finally, choose the first request from H following the starting point next [43].

SimulatedAnnealing$_l$(Sa_l): Construct a Hamiltonian path H visiting each of the requests in the lookahead starting in the current server location and ending in o as follows: obtain H initially by Nn_l or Ins_l. Select initial temperature $T_0 \in \mathbb{R}$ (e.g., $T_0 = 100$), minimum temperature $T_{min} \in \mathbb{R}$ with $T_{min} < T_0$ (e.g., $T_{min} = 5$), maximum number of iterations $L_{max} \in \mathbb{N}$ with unchanged temperature (e.g., $L_{max} = 100$) and set T:=T_0, L:=0. Until $T < T_{min}$, if $L = L_{max}$, then set T:=$0.9 \cdot T$ and L:=1 else set L:=$L + 1$, next choose two requests from H and reverse the order of requests between them to obtain H', and if H' is shorter than H, then set H:=H' else set H:=H' with probability $\exp(-\frac{\Delta}{T})$ where Δ is the difference between the length of H' and the length of H. Finally, choose the first request following the starting point from the shortest Hamiltonian path obtained throughout the procedure next [43].

TabuSearch$_l$(Ts_l): Construct a Hamiltonian path H visiting each of the lookahead points starting in the current server location and ending in o as follows: obtain H initially by Nn_l or Ins_l. Select tabu time $T \in \mathbb{N}$ (e.g., $T = 5$), maximum number D_{max} of diversifications (e.g., $D_{max} = 50$), number of swaps s per diversification (e.g., the largest integer number that is smaller or equal to 30% of the total number of requests included in the Hamiltonian path) and set D:=0. Until $D \geq D_{max}$, carry out the following steps:

1. Choose two requests from H and swap them to obtain H'.
2. If H' is shorter than H and the swapped request pair is not included in the tabu list, set H:=H' and add the swapped request pair to the tabu list with remaining tabu time T; else if H' is shorter than the best Hamiltonian path obtained so far and the swapped request pair is tabu, set H:=H' (aspiration).
3. For all tabu list entries except the new one, decrease the remaining tabu time by one iteration; return to step 1 until all pairs of points have been examined.
4. Perform a random swap in H' for s times (diversification), set H:=H', D:=$D + 1$.

Finally, choose the first request following the starting point from the shortest Hamiltonian path obtained throughout the procedure next [27].

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

Sets and parameters

J	set of requests with $J = \{1, \ldots, N\}$
N	number of requests
c_{ij}	distance between requests $i, j \in J$
c_j^{start}	distance of request $j \in J$ to server start position (which coincides with the current server position)
c_j^{end}	distance of request $j \in J$ to server end position (which coincides with the origin as the server home base)
M	sufficiently large constant (big M)

Variables

$$x_{ij} = \begin{cases} 1 & \text{if request } i \in J \text{ immediately precedes request } j \in J, \\ 0 & \text{else} \end{cases}$$

$$x_j^{start}/x_j^{end} = \begin{cases} 1 & \text{if request } j \in J \text{ is visited first / last,} \\ 0 & \text{else} \end{cases}$$

$T_j \geq 0$ start time of request $j \in J$

Fig. 10. Sets, parameters and variables in the MIP formulation of the Hamiltonian path problem with fixed start and end.

$$\min \quad \sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij} + \sum_{j\in J} c_j^{start}x_j^{start} + \sum_{j\in J} c_j^{end}x_j^{end} \tag{44}$$

$$\text{s.t.} \quad \sum_{j\in J, j\neq i} x_{ij} + x_i^{end} = 1 \qquad i \in J \tag{45}$$

$$\sum_{i\in J, i\neq j} x_{ij} + x_j^{start} = 1 \qquad j \in J \tag{46}$$

$$\sum_{j\in J} x_j^{start} = 1 \tag{47}$$

$$\sum_{j\in J} x_j^{end} = 1 \tag{48}$$

$$T_i + 1 \leq T_j + \mathrm{M}\cdot(1 - x_{ij}) \qquad i, j \in J \tag{49}$$

$$x_{ij}, x_j^{start}, x_j^{end} \in \{0, 1\} \qquad i, j \in J \tag{50}$$

$$T_j \geq 0 \qquad j \in J \tag{51}$$

Fig. 11. MIP formulation of the Hamiltonian path problem with fixed start and end.

OPTIMAL$_l$(OPT$_l$): In Fig. 10, specify N according to the number of seen requests, c according to the distances of the seen requests between each other, c^{start} according to the distances of the requests to the current server position, and c^{end} according to the distances of the requests to the origin. Solve the MIP formulation in Fig. 11. Choose the first request following the starting point in the obtained solution next.

In the MIP formulation in Fig. 11, we decide to get rid of the subtour elimination constraints by establishing precedence relations in Constraint 49 between requests based on decision variables for time instants at which requests are served.

In addition to these algorithms, we also consider their batched versions in order to check whether continuous information release is necessary or whether information release in blocks (so-called batches) of items suffices to obtain satisfactory results. If algorithms operate under batched lookahead, we indicate them with an added suffix B in the algorithm name, e.g., NN$_{10,B}$ means that the NEARESTNEIGHBOR$_{10}$ algorithm is applied once for each batch of 10 requests, and only after all of the 10 requests have been visited a new batch of 10 items is released.

We now present the computational results for the algorithm families NN$_l$, INS$_l$, 2OPT$_l$, 3OPT$_l$, SA$_l$, TS$_l$, OPT$_l$ and their respective batched versions under variable size l of the lookahead set. We select the settings of $n=25$ and $n=100$ requests per sequence; each setting features $m=1000$ independently drawn request sequences. Requests are located in the planar unit square $\mathcal{M} = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ and distance is measured by the Euclidean metric. We discuss results for $n=25$ and refer to [22] for $n=100$. For $n=25$, all lookahead sizes $l \in \{1, 5, 10, \ldots, 25\}$ are tested in order to quantify the value of additional lookahead.

Average results: In contrast to bin packing, the objective value is immediately and heavily affected by the input element processing order and there should be a clear effect of visiting requests in an order different from their release order (rule set substitution).

Fig. 12 confirms the huge benefit of lookahead. Comparing the online with the offline case, reductions of 60.5% (for $\text{Ins}_{l,B}$) up to 67.8% (for Opt_l) are achieved depending on the used algorithm. The marginal benefit of an additional lookahead unit is strictly decreasing for all algorithms. In this sense, provisioning a pure online algorithm with lookahead sets of small size $l \leq 5$ already leads to considerable improvement, whereas for large enough lookahead size $l \geq 20$ only small additional improvement is possible. For instance, already for a lookahead size of $l=5$ the overall tour lengths are reduced to between 57.5% (for Nn_l) and 74.1% (for $\text{Opt}_{l,B}$) of the online tour length. Finally, from the clear separation of the curves marked with dots (e.g., Nn_l) and their respective counterparts marked with rectangles (e.g., $\text{Nn}_{l,B}$) we observe improved behavior under regular request lookahead compared to batched lookahead as a result of the potential for tour length reduction that each additional request in the lookahead brings along.

Overall, lookahead positively affects all algorithms in the same order of magnitude. However, on a detailed look it is also seen that for small lookahead sizes ($l \leq 5$) the simple algorithm Nn_l fares best; sophisticated algorithms such as 3Opt_l, Ts_l and Opt_l exhibit superior performance only for medium to large lookahead, i.e., for ($l \geq 10$). We conclude that a surplus of refined algorithms can only be realized when the overseen time horizon is as large as to guarantee no severe deviations from once computed plans upon new request arrivals. Overall, we also attribute this effect to a reduced probability for zigzagging under Nn_l, albeit returning to previously seen regions cannot be excluded entirely because of future requests.

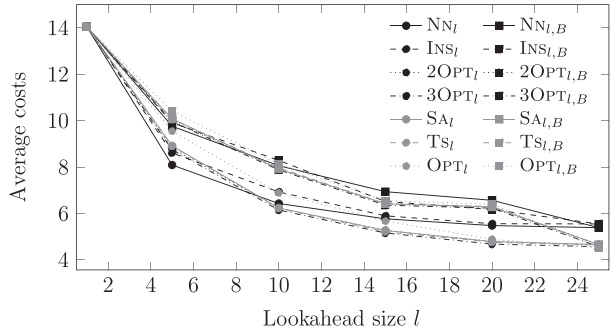

Fig. 12. Average costs for different lookahead sizes and $n=25$ in the TSP.

However, based on further experiments that we conducted we conjecture that the results of an exact reoptimization approach such as Opt_l strongly depend on how the snapshot problem is modeled. In this sense, searching for the "right" snapshot problem is a challenging avenue of future research. Comparing the sophisticated methods among each other, we find that Opt_l is recommendable only in the full lookahead situation ($l=25$), i.e., in the offline problem. For $l \leq 20$ the locally optimal snapshot solutions of Opt_l seem to be unrobust with respect to the integration of new requests. For lookahead sizes $10 \leq l \leq 20$, it is therefore advisable not to use Opt_l, but 2Opt_l, 3Opt_l or Ts_l which are all comparably good in this information regime.

A summary of statistical key figures for the overall tour lengths can be found in Table A8 of Online Appendix A.

Distributional results: Empirical counting distribution functions of objective values and performance ratios are shown exemplary for algorithm classes Nn_l and 3Opt_l in Fig. 13. For successive lookahead levels of small to medium size, the plots of the distribution functions for the total tour length on the left side of Fig. 13 appear clearly segregated from each other. For instance, the curves for lookahead size $l=5$ are already so far away from the curves for lookahead size $l=1$ that the longest tour for $l=5$ is barely longer than the shortest tour for $l=1$. The decrease in the absolute value of tour lengths under lookahead is as substantial as to cause the supports of the density functions related to the given plots not to overlap for $l=1$ and $l=10$. From the decreasing gap between curves of successive lookahead levels for increasing lookahead level, we get a clear confirmation of the decreasing marginal benefit of additional lookahead units that has already been conjectured from the average results. Moreover, the advantage (disadvantage) of Nn_l as a representative of the simple rule-based heuristics over 3Opt_l as a representative of the sophisticated heuristics for $l \leq 10 (l \geq 20)$ becomes apparent by considering the horizontal positions of the curves. For instance, the curve for lookahead level $l=1$ lies farther left for Nn_1 than for 3Opt_1; contrarily, the curve for $l=25$ lies farther right for Nn_{25} than for 3Opt_{25}.

The empirical counting distribution functions of the performance ratios relative to the online versions of the algorithms on the right side of Fig. 13 point towards the huge impact of the first lookahead units and the decreasing marginal benefit of additional lookahead units. Moreover, this illustration allows us to display

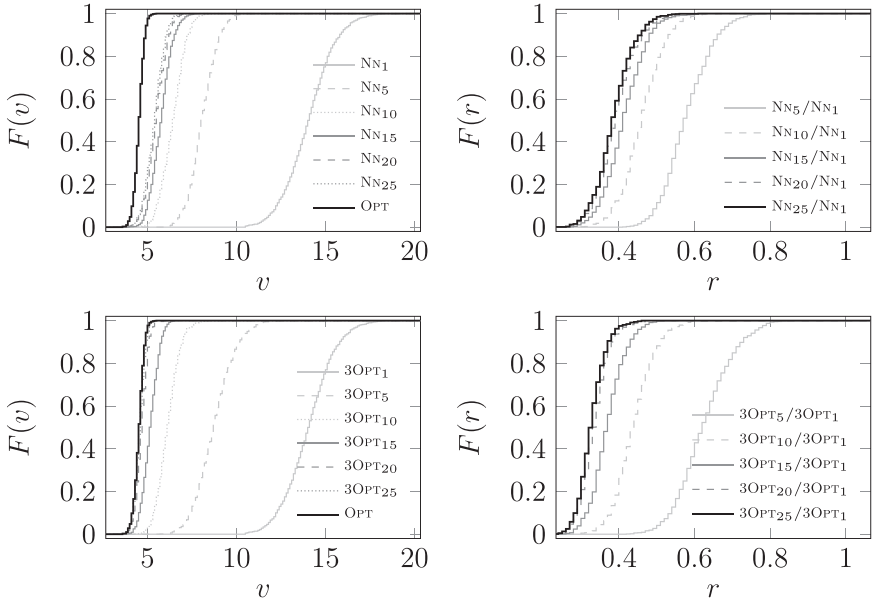

Fig. 13. Empirical counting distribution functions of costs (left) and performance ratios relative to the online case (right) for $n=25$ in the TSP.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

the tour length savings graphically in a fine-grained way. For instance, for lookahead size $l=5$, we immediately see that tour length reductions in the area of 30–40% are probable to occur. Finally, we note that no instance exists for which the provision of lookahead leads to a deterioration in the resulting tour length when compared to the pure online case. Hence, lookahead proves exclusively beneficial.

Observe that for large lookahead sizes $l \geq 20$ all counting distribution functions in Fig. 13 are relatively steep in a characteristic interval which illustrates the homogenous positive effect of additional information on all kinds of request sequences.

Overall, we could identify huge benefits of lookahead in this version of the TSP as a result of the allowance to visit requests in any order. Given this freedom, selecting the right request to be visited next by the right algorithm is substantially rewarded by decreased tour lengths.

A summary of statistical key figures for the performance ratios relative to the online case can be found in Table A9 of Online Appendix A.

5.3. Information pool

Table 3 subsumes the findings of all numerical experiments (including also ski rental, paging and scheduling, cf. [22]) on a granular level in an information pool on observable lookahead effects in different settings.

The column for the total lookahead effect ($\Delta f^{r,r',P,P'}_{ALG,ALG'}$) admits an overall positive effect of lookahead, but also a strong dependency of its magnitude on the problem setting itself. The columns for the partial lookahead effects due to instance revelation rule substitution ($\Delta f^{r,r'}_{ALG}$) and due to rule set substitution ($\Delta f^{P,P'}_{ALG,ALG'}$) additionally illustrate the decomposition of the lookahead effect into an informational and a processual component.

In the TSP and in scheduling with allowed immediate processing, additional lookahead directly paid off upon changing the processing order of input elements because of the direct impact on the objective value. Specifically, the large improvement in the TSP is quantifiable by large tour length reductions of up to 68% (comparing the offline case relative to the pure online case) and all benefits are exclusively attributable to the change in the rule set. In the scheduling experiments, the allowance for immediate processing under lookahead accounts for large savings of up to 75% and nearly all benefits are a result of the change in the rule set. Improvement in negligible order (smaller than 0.1%) is achieved due to the change in the instance revelation rule.

In bin packing, the observed effect was much smaller as a result of the indirect impact of item assignments on the objective value: an item occupies the same bin capacity no matter in which bin it lies and at which time it is packed. Moreover, we found that improvements in packing could only be attained by a clever arrangement of the small objects. Unfortunately, constellations where such improvements can be made are encountered rarely in typical sequences. In total, we observed small savings in the number of required bins of up to 5% comparing the offline case relative to the pure online case. Mainly, this effect consists of the improvement that could be made through the improvement in the instance revelation rule. Taking into account the rule set substitution from “item permutations forbidden” to “item permutations allowed”, we additionally find in bounded-space bin packing that negligible improvement (smaller than 1%) can be attributed to the processual component.

Lookahead in the paging and ski rental problem was also found to have a major positive impact because of the risk-free exploitation of information based on a larger part of the time horizon that is overseen. Here, the additional knowledge allows an algorithm to make a decision that guarantees no future drawbacks. In detail, we observed cost reductions of up to 38% (comparing the offline case relative to the pure online case) in paging with equal probabilities for requested pages, up to 10% when page sequences are generated according to an access graph, and up to 34% in the case of prescribed page frequencies. A large effect of up to 20% cost savings was also observed in the ski rental problem.

The columns $\Delta f^{r,r'}_{ALG}$ and $\Delta f^{P,P'}_{ALG,ALG'}$ show that in ski rental, paging and bin packing merely additional information was responsible for improvements, whereas in the TSP and in scheduling the change of the rule set lead to major improvement. Hence, the ultimate cause for observed lookahead effects also strongly differs between applications.

Column ALG* tells us whether there was an algorithm that could be considered the champion over all lookahead levels. Although this happened rarely, we often observed that heuristics did especially well for small lookahead, whereas exact reoptimization approaches excel heuristics slightly for large lookahead near the offline version.

Column OPT indicates whether exact reoptimization lead to significant improvements: Only minor improvements were observed such that exact reoptimization is no must-have. Although in some problems these methods led to a slight performance enhancement especially for large lookahead, they also offered nearly no benefit for small lookahead when compared to good heuristic reoptimization strategies: optimality of a partial solution often does not migrate to the overall solution because substructures in partial solutions with a positive influence on the objective value are likely to be relinquished during the future solution process.

Table 3
Qualitative summary of the experimental results from [22].

Problem	Type	Attribute	Rule	$\Delta f^{r,r',P,P'}_{ALG,ALG'}$	$\Delta f^{r,r'}_{ALG}$	$\Delta f^{P,P'}_{ALG,ALG'}$	ALG[a]	OPT	Deterioration
Ski rental	Request lookahead	–	–	Large	Large	Zero	Yes	–	No
Paging	Request lookahead	Equal probabilities	–	Large	Large	Zero	No	–	No/yes[a]
		Access graph	–	Medium	Medium	Zero	Yes	–	No/Yes[a]
		Page frequencies	–	Large	Large	Zero	Yes	–	No/Yes[a]
Bin packing	Request lookahead	Classical	Permutations	Small	Small	Zero	No	No	Yes
			No permutations	Small	Small	Zero	No	No	Yes
		Bounded- space	Permutations	Small	Small	Negligible	Yes	No	Yes
			No permutations	Small	Small	Negligible	No	No	Yes
TSP	Request lookahead	–	–	Large	Zero	Large	No	No	Yes
Scheduling	Time lookahead	Single machine	Immediate processing	Large	Negligible	Large			
			No immediate processing	Negligible	Negligible	Zero			Yes
		Parallel machines	Immediate processing	Large	Negligible	Large			
			No immediate processing	Negligible	Negligible	Zero			

[a] Instances with deteriorated objective value in the paging problem were only observed for batching algorithms.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

There was no homogeneous picture about whether additional lookahead can also lead to objective value deterioration or not (column *Deterioration*): in some problems there were rare instances with degraded algorithm performance since lookahead was leading them towards a wrong direction as discovered later in the processing of the input sequence.

Altogether, computational results are affirmative to the exact analysis in [22] concerning the magnitude of the lookahead impact as well as its primarily responsible factors. Our approach of algorithm assessment in the context of a potential provision of lookahead encompassed two stages: in the first stage, an average-case analysis allowed us to find the most promising algorithm candidates for a given lookahead level in terms of expected algorithm behavior. In the second stage, distributional analysis leads to a fine-grained assessment of each candidate's individual risk profile with respect to attainable objective values and performance ratios.

6. Conclusion

Although tremendous research effort has been spent on online optimization over the past two decades, it is still widely believed that the state of the art is yet far from reaching maturity [30]. In particular, there is no agreed groundwork of methods and tools for comprehensive algorithm analysis in online optimization, not to mention in online optimization with lookahead. This paper aimed at contributing towards the elimination of this deficiency.

We first elaborated a clear definition of lookahead in optimization: lookahead is a mechanism of information release that specifies the difference in the process of information disclosure as compared to a reference optimization problem (instance revelation rule substitution) and that might impose a set of constraints differing from the set of constraints in the reference online optimization problem upon the processing of the input elements (rule set substitution).

The general framework serves as a common basis for a domain-independent understanding of the mechanisms and implications of lookahead. A particular emphasis is put on lookahead-related issues such as processing mode, order and accessibility. Hence, devising algorithms for online optimization with lookahead is no longer a problem-specific task independent of a general optimization paradigm, but closely intertwined with the abstract concept of lookahead from Section 2 as well as the framework and classification scheme from Section 3. Solution concepts can now be described in an abstract way using a unified taxonomy.

We emphasize that the terms “online optimization” and “lookahead” are defined in existing literature only in problem-specific contexts of certain publications which prohibits a structurally oriented view on the effects of lookahead. Using the approach outlined in this paper, we were able to attribute a reason to lookahead effects observed in problems and to transfer that knowledge also to other settings (Section 5).

Future research emerges from limitations of the presented approaches and from related topics: we were only concerned with the value of lookahead but not with related costs. Realizing lookahead requires costly technical devices (e.g., RFID, GPS or GIS technology) which facilitate information transmission at an earlier point in time. Installation and operation of such machinery induces a fair amount of costs, and it needs to be checked by economic models whether the benefits exceed the costs of lookahead devices. Another future research branch involves analysis methods for discrete event systems such as (timed) automata, (timed) Markov chains or discrete event simulation and should answer the question of how they can be applied within the framework of online optimization with different types of lookahead, e.g., with time lookahead. Finally, a new research branch could arise from combining and intermixing online optimization with other approaches to regimes of incomplete information like stochastic programming, robust programming or forecast-based optimization.

Appendix A. Supplementary data

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.omega.2015.10.009.

References

[1] Ahlroth L, Schumacher A, Haanpää H. On the power of lookahead in online lot-sizing. Operations Research Letters 2010;38(6):522–526.
[2] Albers S. On the influence of lookahead in competitive paging algorithms. Algorithmica 1997;18(3):283–305.
[3] Albers S. A competitive analysis of the list update problem with lookahead. Theoretical Computer Science 1998;197(1–2):95–109.
[4] Allulli L, Ausiello G, Bonifaci V, Laura L. On the power of lookahead in on-line server routing problems. Theoretical Computer Science 2008;408(2–3):116–128.
[5] Allulli L, Ausiello G, Laura L. On the power of look ahead in on-line vehicle routing problems. In: Wang L, editor. Computing and combinatorics. Berlin, Heidelberg: Springer; 2005. p. 728–736.
[6] Ausiello G, Allulli L, Bonifaci V, Laura L. On-line algorithms, realtime, the virtue of laziness, and the power of clairvoyance. In: Cai J, Cooper S, Li A, editors. Theory and applications of models of computation. Berlin, Heidelberg: Springer; 2006. p. 1–20.
[7] Ausiello G, Crescenzi P, Kann V, Marchetti-Spaccalema A, Gambosi G, Spaccamela A. Complexity and approximation: combinatorial optimization problems and their approximability properties. 2nd ed.. Berlin, Heidelberg: Springer; 2003.
[8] Bellman R. Dynamic programming. Princeton NJ: Princeton University Press; 1957.
[9] Ben-David S, Borodin A. A new measure for the study of on-line algorithms. Algorithmica 1994;11(1):73–91.
[10] Ben-David S, Borodin A, Karp R, Tardos G, Wigderson A. On the power of randomization in on-line algorithms. Algorithmica 1994;11(1):2–14.
[11] Bertsimas D, Brown D, Caramanis C. Theory and applications of robust optimization. SIAM Review 2011;53(3):464–501.
[12] Birge JR, Louveaux FV. Introduction to stochastic programming. 2nd ed.. New York: Springer; 2011.
[13] Borodin A, El-Yaniv R. Online computation and competitive analysis. Cambridge: Cambridge University Press; 1998.
[14] Breslauer D. On competitive on-line paging with lookahead. Theoretical Computer Science 1998;209(1-2):365–375.
[15] Camacho EF, Bordons C. Model predictive control. 2nd ed.. London: Springer; 2007.
[16] Cassandras C, Lafortune S. Introduction to discrete event systems. 2nd ed.. New York: Springer; 2008.
[17] Chung F, Graham R, Saks M. A dynamic location problem for graphs. Combinatorica 1989;9(2):111–131.
[18] Coleman B. Quality vs. performance in lookahead scheduling. In: Proceedings of the 9th international joint conference on information science, 2006. p. 324–7.
[19] Csirik J, Woeginger G. On-line packing and covering problems. In: Fiat A, Woeginger G, editors. Online algorithms: the state of the art. Berlin, Heidelberg: Springer; 1998. p. 147–177.
[20] Dooly D, Goldman S, Scott S. On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM 2001;48(2):243–273.
[21] Dorrigiv R. Alternative measures for the analysis of online algorithms [Ph.D. thesis], University of Waterloo, 2010.
[22] Dunke F. Online optimization with lookahead [Ph.D. thesis], Karlsruhe Institute of Technology, 2014.
[23] Esen M. Design, implementation and analysis of online bin packing problems [Master's thesis], Technische Universität Kaiserslautern, 2000.
[24] Fiat A, Woeginger G. editors. Online algorithms: the state of the art, Springer, 1998.
[25] Garey M, Johnson D. Computers and intractability: a guide to the theory of NP-Completeness. New York: Freeman; 1979.
[26] Ghiani G, Laporte G, Musmanno R. Introduction to logistics systems planning and control. Hoboken, NJ: Wiley; 2004.
[27] Glover F. Tabu search—Part I. ORSA Journal on Computing 1989;1(3):190–206.
[28] Golden B, Raghavan S, Sharda R, Vo S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. Boston, MA: Springer; 2008.
[29] Grove E. Online bin packing with lookahead, in: Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 1995. p. 430–6.
[30] Grötschel M, Krumke S, Rambau J, Winter T, Zimmermann U. Combinatorial online optimization in real time. In: Grötschel M, Krumke S, Rambau J, editors.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009

Online optimization of large scale systems. Berlin, Heidelberg: Springer; 2001. p. 679–704.
[31] Gutin G, Jensen T, Yeo A. Batched bin packing. Discrete Optimization 2005; 2(1):71–82.
[32] Halldórsson M, Szegedy M. Lower bounds for on-line graph coloring. Theoretical Computer Science 1994;130(1):163–174.
[33] Hiller B. Online optimization: probabilistic analysis and algorithm engineering [Ph.D. thesis], Technische Universität, Berlin, 2009.
[34] Imreh C, Németh T. On time lookahead algorithms for the online data acknowledgement problem. In: Kucera L, Kucera A, editors. Mathematical foundations of computer science 2007. Berlin, Heidelberg: Springer; 2007. p. 288–297.
[35] Irani S. Coloring inductive graphs on-line. Algorithmica 1994;11(1):53–72.
[36] Jaillet P, Lu X. Online traveling salesman problems with service flexibility. Networks 2011;58(2):137–146.
[37] Jaillet P, Wagner M. Online routing problems: value of advanced information as improved competitive ratios. Transportation Science 2006;40(2):200–210.
[38] Jaynes E. Information theory and statistical mechanics. Physical Review 1957;106(4):620–630.
[39] Jaynes E. Information theory and statistical mechanics II. Physical Review 1957;108(2):171–190.
[40] Johnson D. Near-optimal bin packing algorithms [Ph.D. thesis], Massachusetts Institute of Technology, 1973.
[41] Kiniwa J, Hamada T, Mizoguchi D. Lookahead scheduling requests for multisize page caching. IEEE Transactions on Computers 2001;50(9):972–983.
[42] Koutsoupias E, Papadimitriou C. Beyond competitive analysis. SIAM Journal of Computing 2000;30(1):300–317.
[43] Lawler E, Lenstra J, Rinnooy Kan A, Shmoys D, editors. The traveling salesman problem: a guided tour of combinatorial optimization. Chichester: Wiley; 1985.
[44] Li W, Yuan J, Cao J, Bu H. Online scheduling of unit length jobs on a batching machine to maximize the number of early jobs with lookahead. Theoretical Computer Science 2009;410(47–49):5182–5187.
[45] Mandelbaum M, Shabtay D. Scheduling unit length jobs on parallel machines with lookahead information. Journal of Scheduling 2011;14(4):335–350.
[46] Mao W, Kincaid R. A look-ahead heuristic for scheduling jobs with release dates on a single machine. Computers and Operations Research 1994; 21(10):1041–1050.
[47] Motwani R, Saraswat V, Torng E. Online scheduling with lookahead: multipass assembly lines. INFORMS Journal on Computing 1998;10(3):331–340.
[48] Pinedo M. Scheduling: theory, algorithms, and systems. 4th ed.. Boston, MA: Springer; 2012.
[49] Puterman M. Markov decision processes: discrete stochastic dynamic programming. Hoboken, NJ: Wiley; 2005.
[50] Shor P. The average-case analysis of some on-line algorithms for bin packing. Combinatorica 1986;6(2):179–200.
[51] Stadtler H, Kilger C, editors. Supply chain management and advanced planning: concepts, models, software, and case studies. 4th ed.. Berlin, Heidelberg: Springer; 2008.
[52] Tinkl M. Online-optimierung der rundreise auf der kreislinie mit informationsvorlauf [Ph.D. thesis], Universität Augsburg, 2011.
[53] Torng E. A unified analysis of paging and caching. Algorithmica 1998; 20(2):175–200.
[54] Yang D, Nair G, Sivaramakrishnan B, Jayakumar H. Round robin with look ahead: a new scheduling algorithm for bluetooth, In: Proceedings of the 2002 international conference on parallel processing workshops, 2002. p. 45–50.
[55] Yeh T, Kuo C, Lei C, Yen H. Competitive analysis of on-line disk scheduling. In: Asano T, Igarashi Y, Nagamochi H, Miyano S, Suri S. editors. Algorithms and computation, Springer, 1996, p. 356–65.
[56] Young N. Competitive paging and dual-guided on-line weighted caching and matching algorithms [Ph.D. thesis], Princeton University, 1991.
[57] Zheng F, Cheng Y, Liu M, Xu Y. Online interval scheduling on a single machine with finite lookahead. Computers and Operations Research 2013;40(1):180–191.
[58] Zheng F, Xu Y, Zhang E. How much can lookahead help in online single machine scheduling. Information Processing Letters 2008;106(2):70–74.

Please cite this article as: Dunke F, Nickel S. A general modeling approach to online optimization with lookahead. Omega (2015), http://dx.doi.org/10.1016/j.omega.2015.10.009