Automated Competitive Analysis of Online Problems
オンライン問題の競合比解析の自動化について

Shingo Masanishi* 正西 申悟 Takashi Horiyama* 堀山 貴史 Kazuo Iwama* 岩間 一雄

{masanisi,horiyama,iwama}@lab2.kuis.kyoto-u.ac.jp

Abstract

An online knapsack problem is an online problem where an online player receives a sequence of items one by one and on every arrival of an item he must decide whether he takes it or not. In an exchangeable online knapsack problem (EOK), the player has two (or more) knapsacks of size 1, and he is allowed to exchange (and also discard) items in his knapsacks. It is known that the competitive ratio of this problem has an upper bound 1.3333 and a lower bound 1.2808 when the number of knapsack $k$ is 2. Unfortunately, the tight bound is not known. The difficulty for obtaining those bounds is mainly caused by a large number of possible cases to be considered in their proofs. In this paper, we propose a computer aided analysis on the competitive ratios of online problems, and show proofs of the upper bounds 1.3660 and 1.333 for 2-bin EOK. These results do not only give an another proof of the known bound, but also give a basis for the tight bound.

Keywords: Online problems, Online algorithms, Competitive ratio, Knapsack problems.

1 Introduction

An *online problem* gives us a sequence of requests and asks to respond immediately to each request without knowing future information. Online problems are a natural topic of interest in many disciplines such as computer science, economics and operations research. Many applications are essentially online, where immediate decisions are required in a real time. Paging in a virtual memory system is perhaps the most studied of such computational problems. Routing in communication network is another well-known application [1, 2]. An algorithm which solves an online problem is called an *online algorithm*. On the other hand, an *offline algorithm* is allowed to respond to each request after receiving all the requests. In other words, an offline algorithm knows the future, while an online algorithm does not. The most famous way to measure the efficiency of online algorithms is a *competitive ratio*, which is the ratio of the cost of an online algorithm to that of the offline algorithm. A good online algorithm makes a competitive ratio 1 as to close.

It is clear that, in many settings, the lack of the information for the future is a great disadvantage. Hence, performances of online algorithms are often much worse than those of offline algorithms, and as a result, the competitive ratio becomes large. Then, it is common to consider a relaxed model of online problems where online algorithms can hold more than one solutions and output the best one. Iwama and Taketomi [3] apply this model to the *removable online knapsack problems* (ROK). They achieved the tight bounds 1.6180 for the one bin model, and 1.3815 for $k$-bin models ($k \geq 2$) where the online player has $k$ bins and he can select one out of the $k$. Recently, a more relaxed model is proposed by the authors, called *an exchangeable online knapsack problem* (EOK) [24]. In this problem, one can move items from one knapsack to another, which is not permitted in ROK. When the player is allowed to have $k$ bins, the problem is called a *$k$-bin EOK* [24]. Although we have shown that the competitive ratio has an upper bound 1.3333 and a lower bound 1.2808 for

*Graduate School of Informatics, Kyoto University. 京都大学情報学研究科.

2-bin EOK, the tight bound is not known. The difficulty for obtaining those bounds is mainly caused by a large number of possible cases to be considered in their proofs.

In this paper, we propose an automated competitive analysis of online problems. There are huge literatue of the computer aided proof so far. The most famous and exciting one is the proof of the four color theorem by Appel and Haken [17, 18]. In their approach, they made a reduction from the theorem to 1,476 subproblems by hand, and then they solved the problems by a computer. Recently, many papers use semidefinite programming to obtain improved analyses of approximation algorithms for various combinatorial optimization problems. Goemans and Williamson [9] were the first to use semidefinite programming for this purpose. They obtained an 0.87856-approximation algorithm for the MAX CUT and MAX 2-SAT problems, and an 0.79607-approximation algorithm for the MAX DI-CUT problem. Feige and Goemans [12] improved the algorithms in [9], and they claimed that the approximation ratio of the algorithms are 0.931 and 0.859 for the MAX-2-SAT and MAX DI-CUT problems, respectively. Krloff and Zwick [10] used extensions of their ideas to obtain an approximation algorithm for MAX 3-SAT with a conjectured performance ratio of $\frac{7}{8}$. This proof is mostly analytic but does rely on calculations carried out in Mathematica.

There are not the computer aided proof of the competitive ratio for online problems. Moreover, too many combinations often makes a proof of a competitive ratio difficult for online problems. So, we decided to carry out the computer aided proof of a competitive ratio for 2-bin EOK.

The rest of the paper is organized as follows. The next section gives the definition of EOK. In Section 3, we propose the computer aided proof for 2-bin EOK. Section 4 is the conclusion.

2 Exchageable Online Knapsack Problem

An Exchangeable Online Knapsack Problem is a variant of a Removable Online Knapsack Problem [3] where the online player can move items from one bin to other. That is, it is permitted that one can exchange items put at once from one knapsack to other. When the online player has k bins, we call the model as k-bin Exchangeable Online Knapsack Problem (k-bin EOK).

Let $B_1, B_2, \ldots, B_k$ denote k bins, where their capacities are always 1. Each B_j has *size* $|B_j|$ which is defined as the total size of the items in B_j. The input and the possible action at round i and the goal are as follows, respectively.

Input : An item u_i, whose size is $|u_i| \in (0, 1]$.

Action : The player decides (1) which bin out of the k bins to put u_i into and (2) which (zero or more) items in the bins (including u_i) are discarded or moved from one bin to other so that $|B_j|$ will be at most 1.0 for all $1 \leq j \leq k$.

Goal : When input is finished, maximize the largest size in the k bins.

Let σ and $\mathcal{A}$ be an *input* sequence $\{u_1, u_2, \ldots, u_i, \ldots, u_n\}$ and an algorithm for k-bin EOK, respectively. For an instance σ, $|\mathcal{A}(\sigma)|$ denotes the *cost* achieved by $\mathcal{A}$, which is defined by the largest size of the bins (i.e., $\max\{|B_1|, \ldots, |B_k|\}$) after the final round for input σ is completed. $|OPT(\sigma)|$ is the cost achieved by the offline optimal algorithm. $\frac{|OPT(\sigma)|}{|\mathcal{A}(\sigma)|}$ is called the *competitive ratio* (*CR*) of algorithm $\mathcal{A}$ for input σ, and its worst-case value, i.e., $CR(\mathcal{A}) = \max_{\sigma} \frac{|OPT(\sigma)|}{|\mathcal{A}(\sigma)|}$, is called *CR* of $\mathcal{A}$.

The related results and our results are shown in Table 1. The tight bounds for the competitive ratio of ROK are shown in [3], and the competitive ratio for 2-bin EOK are shown in [24]. But the tight bounds for 2-bin EOK have not been shown yet.

	Upper Bound	Lower Bound
Online Knapsack Problems	∞	∞
1-bin Removable Online Knapsack Problems	1.6180	1.6180
k-bin Removable Online Knapsack Problems($k \geq 2$)	1.3815	1.3815
2-bin Exchangeable Online Knapsack Problems	1.3333	1.2808

Table 1: Competitive ratio of various Online Knapsack Problems

3 Automated Competitive Analysis

In this section, we show how we prove the upper bounds of *CR* 1.3660 and 1.3333 for 2-bin EOK. We regard the proof as a search problem to check the entire state of the online algorithm. To achieve this, in addition to retrieve the entire state transitions automatically, it is necessary to check the competitive ratio of each state automatically.

3.1 Outline of Algorithm $\mathcal{A}_{system}$

We prove the CR for 2-bin EOK with computer assist. Shown in the section3, this proof makes many states. And one must describe each action, if without computer assist. So, we write the rule for making states to the computer, and make the computer make states. Moreover, each state made by the computer has the inequalities that items held in bins must satisfy. We must analyse whether the cost ratio between offline and online is $< \alpha$, or not in each state. We use Mathematica to analyse the competitive ratio of each state and also the condition for the state transition.

An algorithm which was made to assist the proof is shown in Fig. 1. Let this algorithm be $\mathcal{A}_{system}$. We

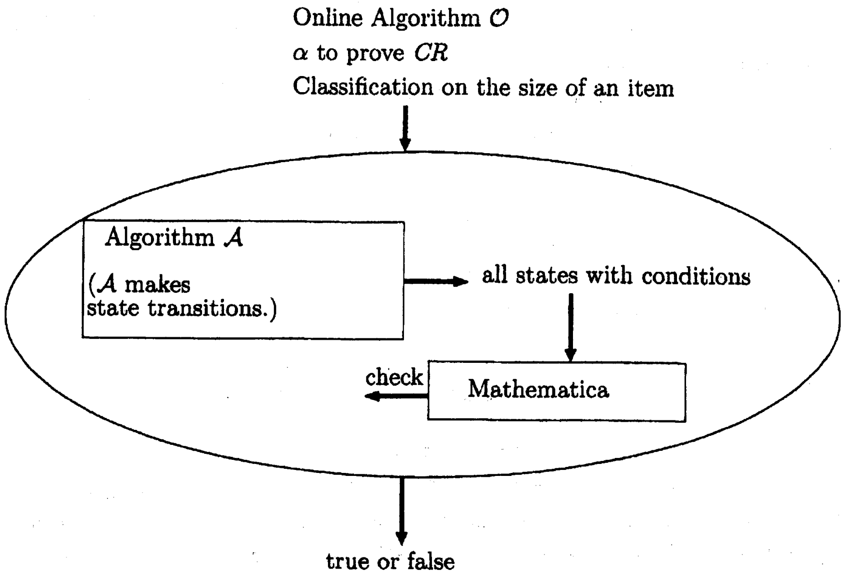

Online Algorithm $\mathcal{O}$
α to prove CR
Classification on the size of an item
Algorithm $\mathcal{A}$
($\mathcal{A}$ makes state transitions.)
all states with conditions
check
Mathematica
true or false

Figure 1: Outline of $\mathcal{A}_{system}$.

give $\mathcal{A}_{system}$ an online algorithm $\mathcal{O}$, α to prove *CR* and a classification on the size of an item as inputs. They have a close relation each other. $\mathcal{O}$ is a primary strategy to make state transitions. Then, $\mathcal{A}_{system}$ processes the problem with inputs and outputs whether it can satisfy $CR < \alpha$, or not.

Next, we describe the action of $\mathcal{A}_{system}$. Firstly, $\mathcal{A}_{system}$ makes state transitions, using inputs and an algorithm $\mathcal{A}$. Each state consists of (W_1, W_2, I) and condition. Their definition is the same in the above argument. $\mathcal{A}$ is a part of $\mathcal{A}_{system}$, and exists to make state transitions. Secondary, $\mathcal{A}_{system}$ checkes whether every state can satisfy $CR < \alpha$, using Mathematica, or not. If every state can satisfy, because it means that we prove $CR < \alpha$ for this problem, $\mathcal{A}_{system}$ outputs true. Otherwise, $\mathcal{A}_{system}$ outputs false.

3.2 Online Algorithm $\mathcal{O}$

3.2.1 Common Strategy

An online algorithm $\mathcal{O}$ is a primary strategy to make state transitions, and given to $\mathcal{A}_{system}$ as an input. Moreover, $\mathcal{O}$, α, and a classification have a close relation each other. Therefore, if $\mathcal{O}$ changes, the others change. In the reverse case, it also does. When $\mathcal{A}_{system}$ makes state transitions, it is very important to determine which items are discarded. $\mathcal{O}$ is designed so that $\mathcal{A}_{system}$ can determine which items are discarded well. That is, we can also say that $\mathcal{O}$ is a primary strategy to determine which items are discarded.

Let t be $\frac{1}{t}$. $\mathcal{O}$ devide items into three diffrent classes S, L, and X. An item in each class is denoted by s, ℓ, and x, respectively. Each item satisfies: $0 < |s| \leq 1-t$, $1-t < |\ell| \leq t$, and $t < |x| \leq 1$.

Now we explain how to devide items. If a player can make a bin larger than the size t, it is clear that CR of this case can satisfy $CR < \alpha$. If a item of X is given at the first time, a player can make the good bin with only x. Therefore, we think only the case where no items of X are given. That is, a class X is designed so that we can ignore the class X as inputs. Next, we discuss a class S. If a ℓ_S must be discarded, Each size of all bins a player has is larger than t. Then, the player satisfies $CR < \alpha$. Therefore, we think also only the case where no items of X are given. That is, $\mathcal{O}$ is designed so that we can think only a class L as inputs and if the player can make the bin larger than size t, the player just discards given items after that.

By the way, we describe a primary principle to determine which items are discarded. The principle is to substitute a smaller item for a larger item. That is, if a player discards a item, the player keeps a smaller item than it to substitute for it. But if a item is by far smaller than the other, it cannot be substituted for a larger one. To substitute v for u, u and v must satisfy the following conditions.

- $|v| \leq |u|$ (Otherwise, the adversary gives an item whose size is $1-|u|$ as an input, and a player cannot hold the item.)
- $\frac{|u|}{|v|} < \alpha$ (Otherwise, the adversary stops inputs. If the size of offline and online are $|u|$ and $|v|$, respectively, a player cannot satisfy $CR < \alpha$.)

That is, if a player discards u, the player must keep v which satisfies the above conditions.

To satisfy the the above substitution conditions, L is devided into more three different classes L_S, L_M, and L_L, shown in Fig.2. An item in each class is denoted by ℓ_S, ℓ_M, and ℓ_L, respectively. Each item satisfies: $1-t < |\ell_S| \leq 1-t^2$, $1-t^2 < |\ell_M| \leq t^2$, and $t^2 < |\ell_L| \leq t$. Because of $\frac{\ell_{Lmax}}{\ell_{Lmin}} < \frac{t}{t^2} = \alpha$, ℓ_{Lmin} can substitute for all ℓ_L's so far given. That is, L_L is a max range which can substitute for an item of size t. A ℓ_M and a ℓ_L cannot be hold in the same bin at the same time, because the total of lower bounds of L_M and L_L is over 1. Moreover, a ℓ_S and a ℓ_M can be hold in the same bin at the same time unconditionaly, because the total of upper bounds of L_S and L_M is under 1.

Furthermore, we focus on whether a group of classes is over t, or not, and regard $2t-1$ and $2-2t$ as bounds to devide more different classes. $((1-t)+(2t-1) = t$ and $2t-1+2-2t)$ In the next section, they are used.

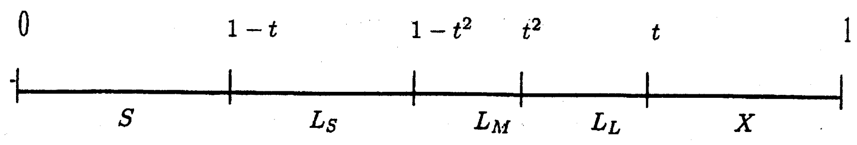


Figure 2: classification of an item

3.2.2 Online Algorithm for $\alpha \approx 1.3660$

Next, we discuss an online algorithm $\mathcal{O}$ where an input α is $\frac{1+\sqrt{3}}{2}(= \alpha \approx 1.3660)$. t is a max point which satisfies $\frac{t}{2-2t} < \frac{1}{t} = \alpha$. That is, t is one of the roots of each equation $2-2t = t^2$ and $2t-1 = 1-t^2$, and t is $\sqrt{3}-1$. Items of each class satisfies: $\frac{3-\sqrt{3}}{2}(\approx 0.2679) < |\ell_S| \leq \frac{\sqrt{3}}{2}(\approx 0.4641)$, $\frac{\sqrt{3}}{2} < |\ell_M| \leq \frac{2-\sqrt{3}}{2}(\approx 0.5359)$, and $\frac{2-\sqrt{3}}{2} < |\ell_L| \leq \sqrt{3}-1(\approx 0.7312)$.

Let $N(c)$ be the max number that a player can hold items of a class c in the same bin at the same time. For exmaple, $N(L_L)$, $N(L_M)$, and $N(L_S)$ are 1, 2, 3, respectively. The number that items of a class c can component the size of offline is at most $N(c)$. Therefore, a player may hold at most $N(c)$ items of a class c at the same time.

In L_L, because $N(L_L)$ is 1 and L_L is a max range which can substitute for an item of size t, a player may hold at least the smallest item of ℓ_Ls so far ever given. In L_M, though $N(L_M)$ is 2, when a player has two ℓ_Ms in a state, the two items cannot be hold in the same bin at the same time. Therefore, the number that items of L_M can component the size of offline is at most 1, and the player may hold at most one item of L_M at the same time. Because of $\frac{\ell_{M max}}{\ell_{M min}} < \frac{t^2}{1-t^2} < \alpha$, a player may also hold at least the smallest item of ℓ_Ms so far ever given. In L_S, For a similar reason, a player may also hold at least the smallest and the second smallest items of ℓ_Ss so far ever given.

After all, a player will substitute a smaller item for a larger item in the same class. That is, if a player must discard an item of a class, the player will discard a larger item of the class and keep a smaller item of the class.

3.2.3 Online algorithm for $\alpha \approx 1.3333$

$t = \frac{1}{\alpha}$ means that CR is improved if t is larger. Therefore, we aims at designing $\mathcal{O}$ which has larger t.

We designed $\mathcal{O}$ which has $t = \frac{3}{4}$. That is, We improved $\mathcal{O}$ so that $\mathcal{O}$ satisfies $CR < \frac{4}{3}(\approx 1.3333)$. The classification of items is shown in Fig. 3. L_M is devided into more two different classes L_{MS} and L_{ML}.

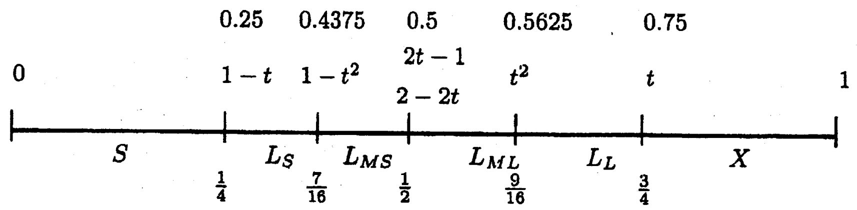


Figure 3: Classification of the items according to their sizes.

Fundamentally a player will substitute a smaller item for a larger item in the same class like the case of

$\alpha \approx 1.3660$. A smallest ℓ_L of ℓ_Ls so far ever given subsitutes for other ℓ_Ls. And ℓ_M can do so. (ℓ_{MS} can substitute for ℓ_{ML}.) But it is not enough. Therefore, we can think to substitute two or more items for one item. For example, To substitute v_1 and v_2 for u, u and v_1 and v_2 must satisfy the following conditions.

- $|v_1| + |v_2| \leq |u|$ (Otherwise, the adversary gives an item whose size is $1 - |u|$ as an input, and a player cannot hold the item.)
- $\frac{|u|}{|v_1|+|v_2|} < \alpha$ (Otherwise, the adversary stops inputs. If the size of offline and online are $|u|$ and $|v_1|+|v_2|$, respectively, a player cannot satisfy $CR < \alpha$.)

We design $\mathcal{O}$ so that two items of L_S substitute for a item of L_L. The subsititution happens when two items of L_S and one item of L_L are held and one or more items must be discarded. If the subtitution happens, $\mathcal{O}$ can satisfy the above second conditions surely. Therefore, $\mathcal{O}$ must satifsy the above first conditions to substitute. That is, when there is a possibility to substitue two ℓ_Ss for one ℓ_L, if $|\ell_S| + |\ell_S| \leq |\ell_L|$ holds, $\mathcal{O}$ discard the largest item of L_L and keep items of L_S , otherwise, discards the largest items of L_S and keep items of L_S.

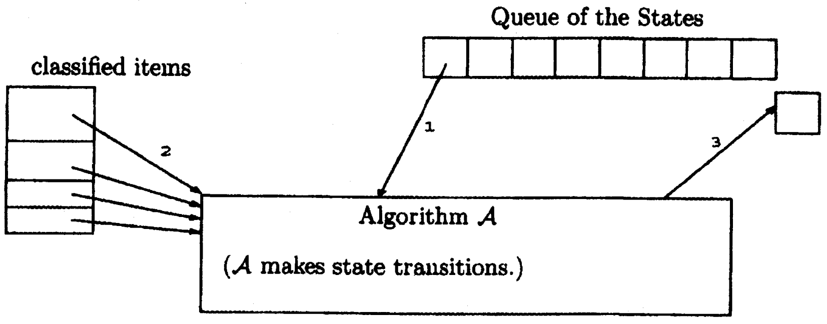
3.3 Algorithm $\mathcal{A}$

3.3.1 Outline of Algorithm $\mathcal{A}$

$\mathcal{A}$ is a part of $\mathcal{A}_{system}$, and exist to make state transitions. When a item is given in a state as an input, it happens a transition from the state. $\mathcal{A}$ outputs the transition state.

As inputs, a state and one of the classified classes which are given to $\mathcal{A}_{system}$ are given to $\mathcal{A}$. Then, $\mathcal{A}$ makes the transition and new conditions which states have, and output the transition state.

The outline of the action of $\mathcal{A}$ is shown in Fig. 4. Because $\mathcal{A}$ is a part of $\mathcal{A}_{system}$, we can say that Fig.



1. input the top of the Queue
2. input the item one by one
3. output the moved states and if the state is new, add the state to the Queue

Figure 4: Outline of an algorithm $\mathcal{A}$.

4 shows a part of actions of $\mathcal{A}_{system}$. There is also a queue of the states in $\mathcal{A}_{system}$. Of course, the queue includes only an empty state at the first time. As inputs, the top of the queue and one of the classified classes which are given to $\mathcal{A}_{system}$ are given to $\mathcal{A}$. (Each classified class is given one by one to one state.) Then, $\mathcal{A}$ makes a transition state, and outputs it. If the state is new, then it is added to the bottom of the queue. The actions are continued till the queue is empty. All states which are made by $\mathcal{A}$ have conditions, and given to Mathematica.

3.3.2 Structure

Next, we discuss the structure of the algorithm $\mathcal{A}$. We know that $\mathcal{A}$ is given one state and one classified item as inputs and makes a transition state which has conditions, using them, from the above argument. Now, we describe how to make a transition state.

Each class has a upper and lower bounds, and let a item of the class be u. $\mathcal{A}$ makes new conditions and a transition state, using W_1, W_2, u, conditions and their bounds. The flow chart of $\mathcal{A}$'s action is shown in Fig. 5. Firstly, $\mathcal{A}$ checkes whether there exists a group which can make the bin more than t with $u \cup W_1 \cup W_2$ unconditionaly or not. If the classification is like Fig. 3, two items of class ℓ_{MS} or each one item of class ℓ_S and ℓ_{ML} constitutes such a group. If there exist, $\mathcal{A}$ outputs EndState. Otherwise, $\mathcal{A}$ goes to the next check. The next check is whether there exists a group which can satisfy both the total of their lowerbound is at least t and the total of their upperbound is at most t, or not. Now, if there exists a group which can make a bin smaller than t, we think the group cannot exceed t. For example. if there exists two ℓ_Ss in Fig. 3, we think $|\ell_S| + |\ell_S| \leq t$. Therefore, if there exists, $\mathcal{A}$ adds the new conditions with the output state and go to the next check. Otherwise, $\mathcal{A}$ goes to the next check without adding the new conditions. For example, two ℓ_S's can do in Fig. 3. The new condition, $\ell_S + \ell_S \leq t$, is added to the output state. The next check is whether there exists a group which can satisfy both the total of their lowerbound is at most 1 and the total of their upperbound is at least 1, or not. If there exists, $\mathcal{A}$ adds the new conditions with the output state and go to the next check. Otherwise, $\mathcal{A}$ goes to the next check without adding the new conditions. For example, three ℓ_S's can do in Fig. 3. The new condition, $\ell_S + \ell_S + \ell_S > 1$, is added to the output state. The next check is whether $\mathcal{A}$ can put u into bins, discarding no items , unconditionaly, or not. For example, the case in which the item of ℓ_S is given in the state3 of Section 3 does. If the player can do, the state which is put the class from the input state is output. Otherwise, $\mathcal{A}$ must discard one or more items. Therefore, $\mathcal{A}$ determines the removed items and output the transition state. Determining the removed items is very important for this system. This action changes with online algorithm $\mathcal{O}$ given as an input.

3.4 Results

We have applied algorithm $\mathcal{A}_{system}$for the two online algorithms of EOK and obtained the following competitive ratios. Figure 6 illustrates the state transtion of the online algorithm of $\alpha \approx 1.3333$ obtained by $\mathcal{A}_{system}$. The state transition and the competitive ratios of each state is confirmed automatically.

Theorem 1 *The online algorithms in Section 3.2.2 and 3.2.3 have the competitive ratios* $\alpha \approx 1.3660$ *and* 1.3333, *respectively.*

4 Concluding Remarks

In this paper, we studied an automated competitive analysis for the exchangeable online knapsack problem. We showed proofs of the upper bounds 1.3660 and 1.333 for 2-bin EOK. Although there is a gap between the lower and the upper bounds, our results give a basis for the tight bound. The bounds for k-bin EOK should be also addressed in the future.

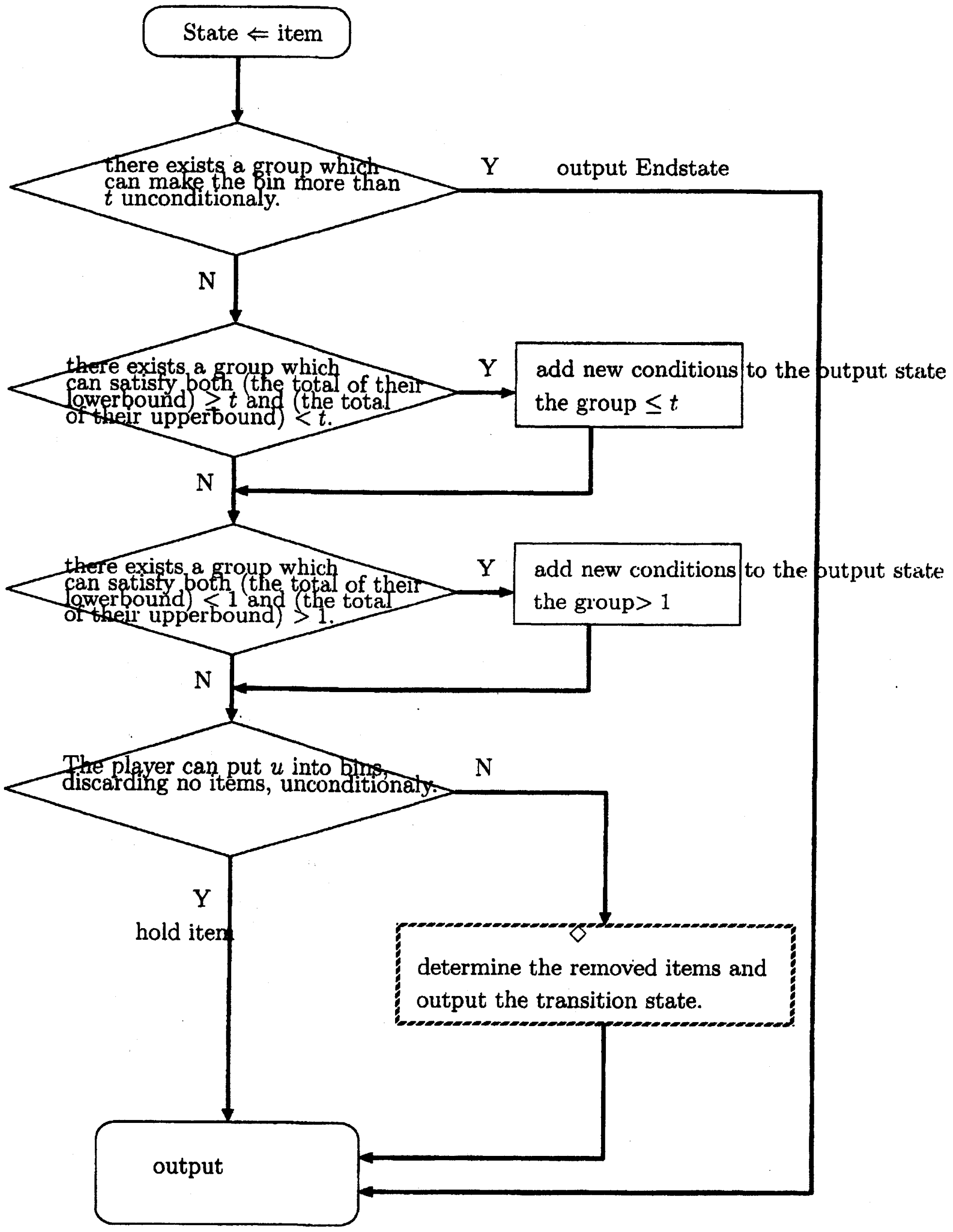

State ⇐ item
there exists a group which can make the bin more than t unconditionaly.
Y
output Endstate
N
there exists a group which can satisfy both (the total of their lowerbound) > t and (the total of their upperbound) < t.
Y
add new conditions to the output state
the group ≤ t
N
there exists a group which can satisfy both (the total of their lowerbound) < 1 and (the total of their upperbound) > 1.
Y
add new conditions to the output state
the group> 1
N
The player can put u into bins, discarding no items, unconditionaly.
N
Y
hold item
determine the removed items and output the transition state.
output

Figure 5: Flow chart of an algorithm $\mathcal{A}$

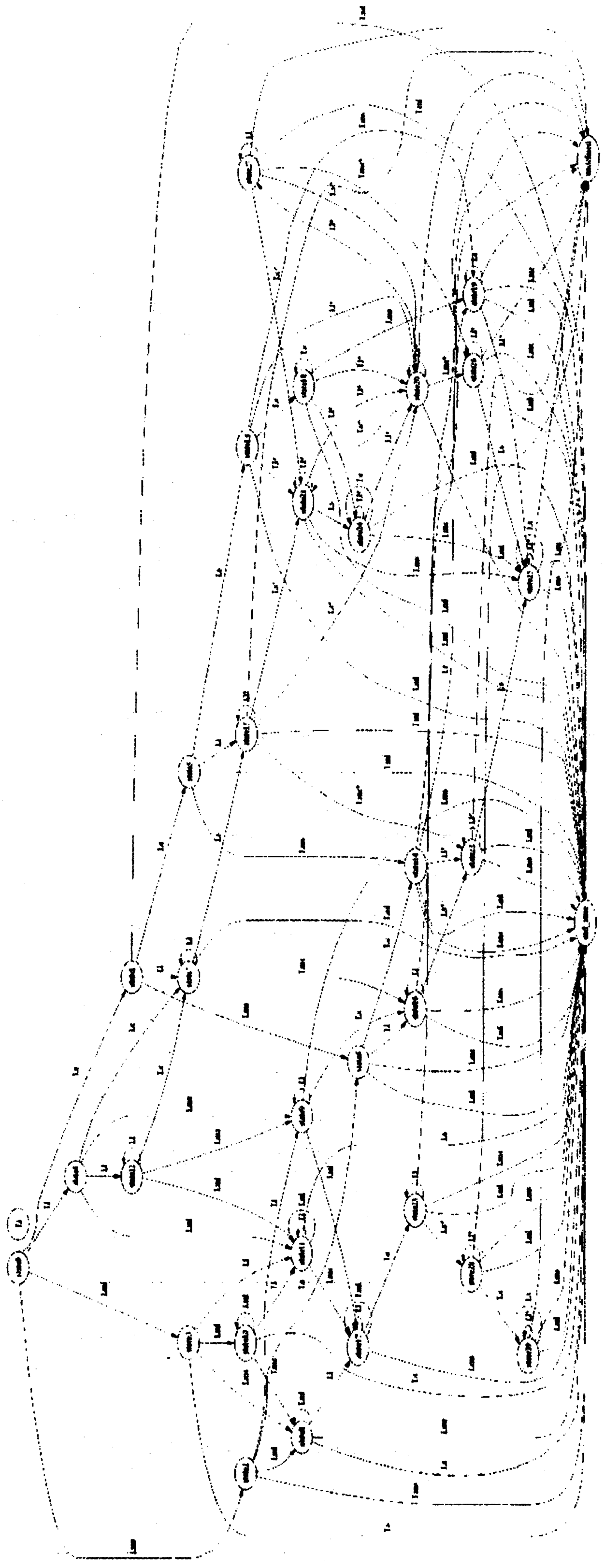


Figure 6: State transition diagram for the online algorithm of $\alpha \approx 1.3333$.

References

[1] A.Borodin and R.El-Yaniv: "Online Computation and Competitive Analysis", Cambridge University Press (1998).

[2] R.M.Karp.: "On-line algorithms versus off-line algorithms, How much is it worth to know the future?", *Proc. IFIP 12th World Computer Congress*, Vol.1, pp. 416–429 (1992).

[3] Kazuo Iwama and Shiro Taketomi: "Removable On-line Knapsack Problems", *Proc. ICALP 2002*, pp.293–305 (2002).

[4] Magnús M and Halldórsson: "Online coloring known graphs", *Electronic J. Combinatorics*, Vol.7, R7 (2000). *http://www.combinatorics.org*

[5] Magnús M, Halldórsson, Kazuo Iwama, Shuichi Miyazaki and Shiro Taketomi: "Online independent set", *Proc. COCOON 2000*, pp.202–209 (2000).

[6] W. Fernandez de la Vega and G. S. Lueker: "Bin packing can be solved within $1+\epsilon$ in linear time", *Combinatorica*, Vol. 1(4), pp. 349–355 (1981).

[7] G. Gambosi, A. Postiglione and M. Talamo: "Algorithms for the relaxed online bin-packing model", *SIAM Journal on Computing*, Vol. 30(5), pp. 1532–1551 (2000).

[8] Uri Zwick: "Computer assisted proof of optimal approximability results", *Proc. SODA*, pp.496–505 (2002).

[9] Michel X. Goemans and David P. Williamson: "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming", *Journal of the ACM (JACM)*, Vol.42, No.6, pp.1115–1145 (1995)

[10] H. Karloff and U. Zwick: "A 7/8-approximation algorithm for MAX 3SAT?", *Proc. FOCS, Miami Beach, Florida*, pp. 406–415 (1997).

[11] Uri Zwick: "Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint", *Proc. SODA* , p.201–210 (1998)

[12] U. Feige and M. X. Goemans: "Approximating the value of two prover proof systems, with applications to MAX-2SAT and MAX-DICUT", *Proc. ISTCS*, Tel Aviv, Israel, pp.182–189 (1995).

[13] George S. Lueker : "Average-case analysis of off-line and on-line knapsack problems", *Proc. SODA*, pp.179–188 (1995).

[14] A. Marchetti-Spaccamela and C. Vercellis: "Stochastic on-line knapsack problems", *Mathematical Programming*, Vol.68, No1, pp.73–104 (1995).

[15] D. D. Sleator and R. E. Tarjan: "Amortized Efficiency of List Update and Paging Rules", *Communications of the ACM*, Vol.28, No.2, pp.202–208 (1985).

[16] D. D. Sleator and R. E. Tarjan: "Self-Adjusting Binary Search Trees", *Journal of the ACM*, Vol.32, No.3, pp.652–686 (1985).

[17] K.Appel and W.Haken: "Every planar map is four colorable . Part 1.Dischargin", *Illinois Journal of Mathematics*, Vol21, pp.429–490 (1977).

[18] K.Appel and W.Haken and J.Koch: "Every planar map is four colorable. Part 2. Reducibility", *Illinois Journal of Mathematics*, Vol21, pp.491–597 (1977).

[19] A. Marchetti-Spaccamela and C. Vercellis: "Stochastic on-line knapsack problems", Manuscript.

[20] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, Vol.3 No.4, pp.299–325 (1974).

[21] S. S. Seiden, "On the online bin packing problem," *Proc. ICALP 2001*, pp.237–248 (2001).

[22] A. van Vliet, "On the asymptotic worst case behavior of harmonic fit," *J. Algorithms*, Vol.20, pp.113–136, (1996).

[23] J. Csirik, G. Galambos, and G. Turán, "A lower bound on on-line algorithms for decreasing lists," *Proc. EURO VI*, (1984).

[24] S. Masanishi and K. Iwama, "Online Knapsack Problem for Exchangeable Items" *Tech. Report of IEICE*. Vol.103, No.31, COMP2003-1, pp.1–8 (2003).