Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 154 (2006) 2247–2262

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

How to collect balls moving in the Euclidean plane☆

Yuichi Asahiro[a], Takashi Horiyama[b], Kazuhisa Makino[c], Hirotaka Ono[d], Toshinori Sakuma[e], Masafumi Yamashita[d]

[a]*Department of Social Information Systems, Faculty of Information Science, Kyushu Sangyo University. 2-3-1 Matsukadai, Higashi-ku, Fukuoka 813-8503, Japan*

[b]*Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan*

[c]*Department of Mathematical Informatics, Graduate School of Information and Technology, University of Tokyo, Tokyo 113-8656, Japan*

[d]*Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 812-8581, Japan*

[e]*Toshiba Solutions Corporation, Toshiba Building 1-1, Shibaura 1-chome, Minato-ku, Tokyo 105-6691, Japan*

Received 4 June 2004; received in revised form 2 November 2004; accepted 21 September 2005

Available online 25 July 2006

Abstract

In this paper, we study how to collect n balls moving with a fixed constant velocity in the Euclidean plane by k robots moving on straight track-lines through the origin. Since all the balls might not be caught by robots, differently from Moving-target TSP, we consider the following 3 problems in various situations: (i) deciding if k robots can collect all n balls; (ii) maximizing the number of the balls collected by k robots; (iii) minimizing the number of the robots to collect all n balls. The situations considered in this paper contain the cases in which track-lines are given (or not), and track-lines are identical (or not). For all problems and situations, we provide polynomial time algorithms or proofs of intractability, which clarify the tractability–intractability frontier in the ball collecting problems in the Euclidean plane.

Keywords: Moving-target TSP; Vehicle routing problem; Partially ordered set; Combinatorial optimization

1. Introduction

Let us consider a general setting for the problem of scheduling n jobs $J_1, J_2, \ldots, J_n$ for k identical machines $M_1, M_2, \ldots, M_k$. Each job J_i is nonpreemptive and requires p_i time to process. A most popular setting assumes the parameters p_i to be constants [14]. Other recent works considered deteriorating jobs, where p_i may vary depending on the actual start time t, which is typically given by a linear function $p_i(t) = a_i - tb_i$. Here a_i models a basic processing time, while b_i a learning effect [4,15].

Another effect of the job execution history H of a machine is captured as a setup time $d_i(H)$, which defines the processing time of the job processed next on the machine [19]; that is, $p_i(H) = a_i + d_i(H)$ for job J_i, provided that it is executed on a machine with history H. The setup time d_i can depend on the whole history, but may be natural to

☆ This work was partially supported by the Scientific Grant-in-Aid by the Ministry of Education, Science, Sports and Culture of Japan. A preliminary version appeared in [2].

E-mail addresses: asahiro@is.kyusan-u.ac.jp (Y. Asahiro), horiyama@i.kyoto-u.ac.jp (T. Horiyama), makino@mist.i.u-tokyo.ac.jp (K. Makino), ono@csce.kyushu-u.ac.jp, mak@csce.kyushu-u.ac.jp (H. Ono), Sakuma.Toshinori@toshiba-sol.co.jp (T. Sakuma).

0166-218X/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.dam.2006.04.020

assume to depend only on the previous job J_h. Let d_{hi} be the setup time necessary to start job J_i after J_h. When d_{hi} is a constant for any pair of jobs J_h and J_i, the problem is essentially equivalent to the classical travelling salesman problem (TSP) for a single machine, and to the vehicle scheduling problem (VSP) for multiple machines, where we associate the jobs J_i and the processing time $a_i + d_{hi}$ with the cities and the distances between a pair of cities, respectively.

If d_i depends more on the history and does not only on the previous job but also on the actual start time t, we need to consider a TSP or a VSP variant where the locations of cities may dynamically change. This kinetic variation of TSP has been studied under the names of moving-target TSP [10] and kinetic TSP [9], which motivates our work.

We consider the problem of collecting n balls (or objects) moving around by using k robots. We often encounter this kind of situation as real-world applications. For example, manipulators gather a set of components flowing on belt-conveyers in an assemble line, fishing boats pursuit to catch schools of fishes in the sea, and supply aircrafts resupply patrolling aircrafts in the sky.

The problem can however also be seen as a natural extension of moving-target TSP and kinetic TSP. In the classical TSP, we are given a set of stationary balls (or cities) associated with the distances between each pair of balls. The objective is to compute a shortest tour that collects all the balls by a robot (or visits all the cities by a salesman). In the Moving-Target TSP, the balls collected by a robot are not stationary but moving around. Clearly, Moving-Target TSP is NP-hard, since classical TSP is a special case of Moving-Target TSP in which the velocities of all objects are permanently 0.

Helvig et al. [10] investigated this problem when each ball b_i moves at a given constant velocity $\mathbf{v}_i$ from an initial position $p_i \in \mathbb{R}^d$, and a robot starts at the origin with the maximum speed $v > |\mathbf{v}_i|$. They showed that the problem can be solved in $O(n^2)$ time if $d = 1$ (i.e., the robot and all balls are confined to a single line), and can be approximated in polynomial time with ratio $(1 + \alpha)$, where α is an approximation ratio of an arbitrary classical TSP heuristic, if $O(\log n / \log \log n)$ balls are moving. They also considered the case in which multi-robots can be used. Hammar and Nilsson [9] studied the two-dimensional Euclidean case. For example, it can be shown that, if all the balls move with the same velocity, the problem has a PTAS, but in general, it cannot be approximated better than by a factor of $2^{\Omega(\sqrt{n})}$ in polynomial time, unless P = NP.

In the moving target TSP (or kinetic TSP), it is assumed that any robot can move faster than any ball, and hence robots can always collect all balls. However, in some scenarios (e.g., fishing boats) it is natural to consider the case in which robots cannot collect all balls. This situation results from the fact that robots cannot move faster than balls and/or robots can move only in the restricted spaces such as lines. In this situation, the following fundamental problems have to be studied:

(i) Deciding if given k robots can collect given n balls, and if yes, computing such a robot schedule.
(ii) Computing a robot schedule that maximizes the number of the balls collected by given k robots.
(iii) Computing a robot schedule that minimizes the number of the robots collecting given n balls.

It is clear that (ii) and (iii) are natural extensions of (i). We generically call these kinds of problems the *ball collecting problems* (*BCPs*).

Each of the problems can be easily seen to be NP-hard when arbitrary moves are allowed both for balls and robots. (Indeed, Euclidean Hamiltonian path problems, for example, can be reduced to problem (i) under a certain assumption.) Thus, we identify the goal of this paper to give a complete picture of the tractability–intractability frontier in the BCP. This paper investigates the BCPs in the Euclidean plane, under the assumption that each of the balls and the robots can move only on a straight line (and show the results given in Table 1). More formally, our problems can be defined as follows.

Given a set $B = \{b_1, b_2, \ldots, b_n\}$ of balls in the Euclidean plane (with x- and y-axes), each b_i moving at a constant velocity $\mathbf{v}_i = (v_i \cos \varphi_i, v_i \sin \varphi_i)$ from an initial point $p_i^{(0)} = (x_i^{(0)}, y_i^{(0)}) \in \mathbb{R}^2$, and a set $R = \{r_1, r_2, \ldots, r_k\}$ of (homogeneous) robots, each r_j starting from the origin (or depot) $o = (0, 0)$ and moving with maximum speed $v > 0$ on a line ℓ_j through the origin o, compute robot schedules described in (i), (ii) and (iii).

Fig. 1 illustrates our problems, when 2 robots move on 2 different lines. We also consider the case in which lines ℓ_j are not predetermined (i.e., we can decide an angle θ_j of line ℓ_j as a part of the scheduling).

Here we emphasize that our most restrictive problems when n robots all move on a given single line ℓ still also have practical applications: they can be regarded as new models of the *VSP* on line-shaped network (e.g., [11,16,3,18]).

Table 1
Summary of results

Lines ℓ_j		(i)			(ii)		(iii)
		$k=1$	$k=2$	$k\geqslant 3$	$k=1$	$k\geqslant 2$	
Input	single	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n^3)$	$O(n\log n)$
	multiple	$O(n\log n)$	$O(n^2)$	NP-complete	$O(n\log n)$	NP-hard	NP-hard
Output	single	$O(n^2\log n)$	$O(n^3\log n)$	$O(n^3\log n)$	$O(n^3\log n)$	$O(n^5)$	$O(n^3\log n)$
	multiple	$O(n^2\log n)$	$O(n^6)$	NP-complete	$O(n^3\log n)$	NP-hard	NP-hard

Input and Output mean that lines ℓ_j are chosen as a part of input and output, respectively.

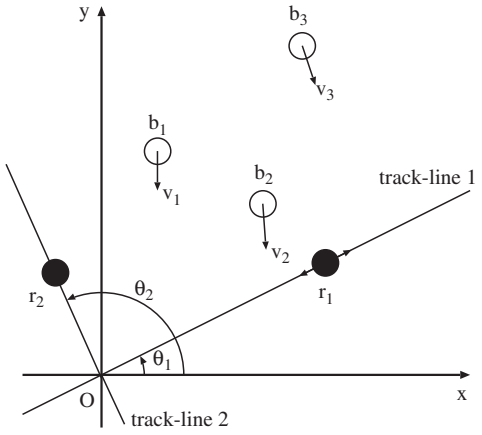

Fig. 1. BCP with 2 robots on 2 track-lines.

VSPs are classified by the conditions on release times, handling times, due times (or deadlines), and the objectives such as minimizing the completion time and minimizing the maximum lateness. Robots and catching balls in our case are regarded as vehicles and jobs, respectively. We consider only the balls which crosses line ℓ exactly once (see the discussion in Section 2.1). The handling time of each job is defined as 0, and the release time and the due time of job b_i are defined as the time when it crosses line ℓ. Each job b_i is located at the position where it crosses line ℓ. Then we can see that problem (i) is to decide whether the maximum lateness of this VSP is 0. This kind of setting is well studied. However, the objectives corresponding to problems (ii) and (iii) (for example, to compute a schedule of vehicles that maximizes the number of the jobs done by their due times) are not investigated as the VSPs.

We extensively study the BCPs defined above mainly from computational points of view. For all problems, polynomial time algorithms or proofs of intractability are provided (although it remains important to improve efficiency of the algorithms for polynomially solvable cases and to develop polynomial approximation algorithms for NP-hard cases). Thus, our work gives a complete picture of the tractability–intractability frontier in the two-dimensional Euclidean BCP when balls b_i move with fixed constant velocities $\mathbf{v}_i$ and homogeneous robots r_j move lines ℓ_j through the depot o. A summary of the results in this paper is given in Table 1.

We first consider the case in which n robots all move on a given single line ℓ, i.e., $\ell_j=\ell$ for all j. In this case, we present an $O(n\log n)$ time algorithm for the problem (iii) (and hence (i) is solvable in $O(n\log n)$ time). This is optimal in the sense that the problem contains the sorting problem as a special case, where the sorting problem is known to need $\Omega(n\log n)$ time for the comparison-based algorithms. Moreover, we show that the problem (ii) is solvable in $O(n\log n)$ time, if $k=1$, and $O(n^3)$ time if $k\geqslant 2$. The result when $k=1$ is also optimal in the sense above.

On the other hand, if given lines ℓ_j are not identical, most problems seems to be intractable. Note that the case of $k=1$ is studied in the single line case. We show that the problem (i) can be solved in $O(n^2)$ time when $k=2$, however, it is NP-complete when $k\geqslant 3$. The problems (ii) and (iii) are both NP-hard.

We finally study the case in which lines ℓ_j are not given (i.e., we can decide an angle θ_j of line ℓ_j as a part of the scheduling). We show that $\Theta(n^2)$ lines are essentially different for our problems, although there exist infinitely many lines through the origin. By combining this with the previous positive results, we provide polynomial time algorithms. For example, if we choose a single line ℓ for all robots, then the problem (iii) is solvable in $O(n \log n) \times O(n^2) = O(n^3 \log n)$ time, and if we can choose distinct lines for each robot, the problem (i) when $k = 2$ can be solved in $O(n^2) \times \binom{O(n^2)}{2} = O(n^6)$ time. We show that the problem (i) when $k = 1$ can be solved in $O(n^2 \log n)$ time, which improves upon the simple combined algorithm. Furthermore, we show the intractability of the problems does not depend on the fact that lines are given or not.

The rest of the paper is organized as follows. In Sections 2 and 3, we investigate the BCPs when the track-lines ℓ_j are given in advance. Section 2 deals with the case in which all ℓ_j are identical, and Section 3 deals with the general case. Section 4 considers the BCPs when ℓ_j are not given. Section 5 concludes this paper and discusses some possible extensions of the BCPs.

2. BCPs with a given single track-line

In this section, we consider the BCPs when all k robots move on the identical line ℓ.

2.1. Single robot BCP

We start with the most fundamental situation that a single robot tries to collect all n balls. We first exclude two kinds of balls: (1) the balls which do not cross the track-line ℓ and (2) the balls which move on ℓ. Note that the robot never catches the balls of type (1). Let b_i be a ball of type (2). If b_i goes away from the origin with the speed $v_i > v$, then the robot cannot catch up with it; otherwise, it is always caught by the robot. Hence we assume without loss of generality that all balls cross ℓ exactly once.

Let t_i denote the time when ball b_i crosses the line ℓ, and let $p_i = (x_i, y_i)$ denote the point where b_i crosses ℓ. Define $d_i = dist(o, p_i)$ if $x_i \geqslant 0$, and $d_i = -dist(o, p_i)$, otherwise, where $dist(o, p_i)$ denotes the Euclidean distance between the origin o and p_i.[1] It is clear that if the robot can catch all balls, then the balls have to be caught in the order of t_i. Therefore, problem (i) can be solved by first sorting t_i $(i = 1, \ldots, n)$ and then checking if

$$v(t_{p_{k+1}} - t_{p_k}) \geqslant |d_{p_{k+1}} - d_{p_k}|, \quad k = 1, 2, \ldots, n-1,$$

where $t_{p_1} \leqslant t_{p_2} \leqslant \cdots \leqslant t_{p_n}$. This immediately implies the following result.

Theorem 1. *It can be checked in* $O(n \log n)$ *time if all balls can be collected by a robot moving on a given track-line. Furthermore, if so, such a robot schedule can be computed in* $O(n \log n)$ *time.*

In the following sections, we show that problem (ii), which is a generalization of problem (i), is also solvable in $O(n \log n)$ time. This is optimal in the sense discussed in the introduction.

Before describing an optimal algorithm, we first present a simple $O(n^2)$ time algorithm for the problem (ii) in order to understand the optimal one easily. The algorithm makes use of a directed acyclic graph (DAG) constructed from d_i and t_i above.

2.1.1. DAG based algorithm

Let θ denote the angle of a track-line ℓ. A directed graph $G(\theta) = (V = \{0, 1, 2, \ldots, n\}, A = \bigcup_{i=0}^{n} A_i)$ is defined by

$$A_0 = \{(0, i) \mid \text{The robot starting at the origin } o \text{ can catch ball } b_i\},$$

$$A_i = \{(i, j) \mid \text{The robot can catch ball } b_j \text{ after catching } b_i\}, \quad i = 1, 2, \ldots, n.$$

It is easy to see that $(0, i) \in A_0$ if and only if $v \cdot t_i \geqslant |d_i|$, and $(i, j) \in A_i$ for $i \geqslant 1$ if and only if $v \cdot (t_j - t_i) \geqslant |d_j - d_i|$. For example, Fig. 2 shows the situation that all balls can be caught directly by the robot starting at the origin o (since

[1] If ℓ coincides with y-axis, $d_i = dist(o, p_i)$ if $y_i \geqslant 0$, and $d_i = -dist(o, p_i)$ otherwise.

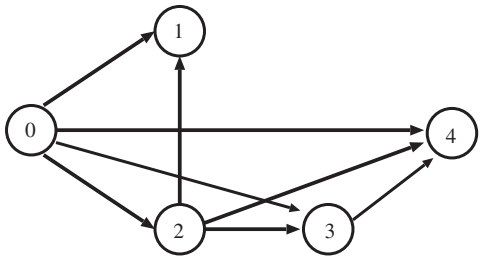

Fig. 2. An example of a directed graph $G(\theta)$.

$G(\theta)$ contains arcs (0, 1), (0, 2),(0, 3) and (0, 4)), ball b_1 can be caught after catching b_2 (since $G(\theta)$ contains arc (2, 1)), but b_3 cannot be caught after catching b_1 (since $G(\theta)$ does not contain arc (1, 3)), and so on.

Note that $G(\theta)$ is transitive, i.e., if $(i_1, i_2), (i_2, i_3) \in A$ then $(i_1, i_3) \in A$. If $G(\theta)$ contains a cycle C, then C does not contain node 0, since no arc enters node 0, and all nodes in C have the same d_i and t_i (i.e., $d = d_i$ and $t = t_i$ for all $i \in C$), since $i, j \in C$ implies $(i, j), (j, i) \in A$ by the transitivity of A, which further implies $t_j - t_i, t_i - t_j \geqslant |d_j - d_i|/v\ (\geqslant 0)$. Moreover, we can see the following proposition.

Proposition 1. *Any directed path from node* 0 *in* $G(\theta)$ *represents feasible robot schedules. Conversely, any feasible robot schedule can be represented by a directed path from node* 0. *Furthermore, the length of the path in* $G(\theta)$ *represents the number of the balls collected under the corresponding schedules.*

We now construct the DAG $G^*(\theta) = (V^*, A^*)$ from $G(\theta)$ by contracting every strongly connected component in it. Here node 0^* in V^* corresponding to 0 in V has a weight $w(0^*) = 0$, and each node $i^*(\neq 0^*)$ in V^* has a weight $w(i^*)$ representing the number of the nodes in V contracted into i^*. Then the problem (ii) (i.e., maximizing the number of the collected balls) can be solved by finding a longest (or critical) path starting from 0^* (corresponding to 0 in $G(\theta)$) in this DAG $G^*(\theta)$, where the length of a path P is the weighted sum of the nodes in P. This problem is also studied in PERT as one of the scheduling problems, and is solvable in $O(|V^*| + |A^*|)$ time (e.g., [12]). Since $|V| \leqslant n$, $|A^*| \leqslant n^2$, and $G^*(\theta)$ can be constructed in $O(n^2)$ time, the problem (ii) is solvable in $O(n^2)$ time.

Theorem 2. *We can compute in* $O(n^2)$ *time a robot schedule that maximizes the number of the balls collected by a robot moving on a given track-line.*

One might see that the size of $G^*(\theta)$ is small if it is constructed from a problem instance of the BCP. Unfortunately, this is not the case. We have a problem instance such that $|A^*| = \Omega(n^2)$. More precisely, even if we remove from A^* all transitive arcs a (i.e., $a = (i_1, i_3)$ if $(i_1, i_2), (i_2, i_3) \in A^*$), the size of arcs is still $\Omega(n^2)$. This means that all the algorithms constructing G^* explicitly require $\Omega(n^2)$ time. Therefore, we need a method to represent G^* (or G) implicitly, in order to construct a faster algorithm.

2.1.2. Chart based algorithm

This section introduces a chart $H(\theta)$ to represent $G(\theta)$ implicitly, and based on this chart, we present an $O(n \log n)$ time algorithm for the problem (ii) when a single robot moves on a given track-line. We first consider a two-dimensional chart $F(\theta)$ with t- and d-axes. $F(\theta)$ has $n + 1$ points (t_i, d_i), $i = 0, 1, \ldots, n$, associated with the region R_i defined by the following 2 inequalities:

$$d \leqslant v(t - t_i) + d_i,$$

$$d \geqslant -v(t - t_i) + d_i.$$

Here $(t_0, d_0) = (0, 0)$, representing the origin o in the (original) Euclidean plane $\mathbb{R}^2$. Fig. 3 shows an example of $F(\theta)$. In this chart, a point (t, d) represents a state that a robot is located at point d at time t. Therefore, a point (t, d) is contained in R_i if and only if a robot starting at state (t_i, d_i) can reach the state (t, d). For example, $(t_1, d_1) \in R_2 \subseteq R_0$ in Fig. 3 means that robot can catch ball b_1 after moving from the origin at time 0 and even after catching ball b_2.

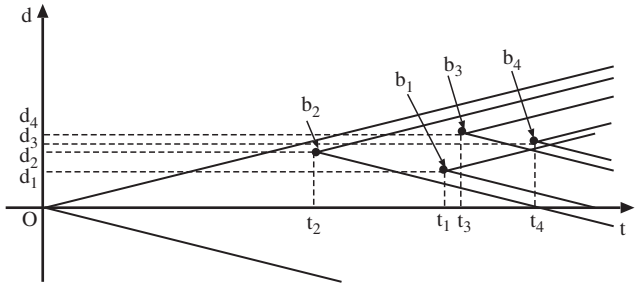

Fig. 3. A chart $F(\theta)$ corresponding to the directed graph $G(\theta)$ in Fig. 2.

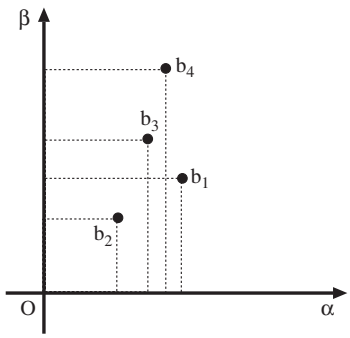

Fig. 4. A chart $H(\theta)$ corresponding to $F(\theta)$ in Fig. 3.

We now define a chart $H(\theta)$ with α- and β-axes to understand this relations easily. For each i, we define α_i and β_i by $\alpha_i = (vt_i - d_i)/(2v)$ and $\beta_i = (vt_i + d_i)/(2v)$; t values of the points at which lines $d = -vt$ (resp., $d = vt$) and $d = v(t - t_i) + d_i$ (resp., $d = -v(t - t_i) + d_i$). Chart $H(\theta)$ has $n + 1$ points (α_i, β_i), $i = 0, 1, \ldots, n$. Here, we identify the points (α_i, β_i) $(i = 1, 2, \ldots, n)$ with balls b_i. Fig. 4 shows an example of $H(\theta)$ (corresponding to $F(\theta)$ in Fig. 3).

For two points (α_i, β_i) and (α_j, β_j), we write $(\alpha_i, \beta_i) \leqslant (\alpha_j, \beta_j)$ if $\alpha_i \leqslant \alpha_j$ and $\beta_i \leqslant \beta_j$, and $(\alpha_i, \beta_i) < (\alpha_j, \beta_j)$ if $(\alpha_i, \beta_i) \leqslant (\alpha_j, \beta_j)$ and $(\alpha_i, \beta_i) \neq (\alpha_j, \beta_j)$.

Lemma 1. *Let α_i and β_i be defined as above. Then we have the following necessary and sufficient conditions.*

(i) *Ball b_i can be caught by a robot starting at origin o if and only if $(0, 0) \leqslant (\alpha_i, \beta_i)$ holds.*
(ii) *Ball b_j can be caught by a robot after catching ball b_i if and only if $(\alpha_i, \beta_i) \leqslant (\alpha_j, \beta_j)$ holds.*

Corollary 1. *Given a chart $H(\theta)$, we can construct $G(\theta)$ (and hence $G^*(\theta)$) by connecting arcs from i to j if $(\alpha_i, \beta_i) \leqslant (\alpha_j, \beta_j)$.*

For example, we can see that Fig. 2 can be constructed from Fig. 4. Our algorithm uses $H(\theta)$ instead of $G^*(\theta)$. Before showing how to use this H, let us consider the dual of the problem (ii).

Note that a transitive DAG $G^*(\theta) = (V^*, A^*)$ can be regarded as a partially ordered set (*poset*, in short) $P(\theta) = (V^*, \preccurlyeq)$, where $i^* \preccurlyeq j^*$ if either $i^* = j^*$ or $(i^*, j^*) \in A^*$. Let us recall that a *chain* (resp., *antichain*) in a poset is a set of pairwise comparable (resp., incomparable) elements. Then the following theorem is well-known, called *a weighted version of the polar* (*or dual*) *Dilworth theorem* [13].

Theorem 3. *Let P be a finite poset with a nonnegative integer weight $w(e)$ of each element e. Then the maximum weight of a chain is equal to the minimum number of antichains covering all elements e in P by $w(e)$ times.*

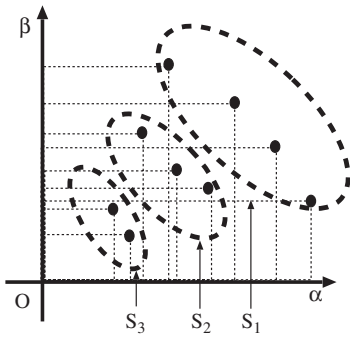

Fig. 5. Algorithm ANTICHAIN-PARTITION.

It follows from Proposition 1 and the subsequent discussion that the problem (ii) is to compute a maximum weighted chain of P. Therefore, by Theorem 3, the dual of the problem (ii) is to compute a minimum w-multiple antichain covering of P. We construct an optimal algorithm for the problem (ii) by solving this dual problem with the help of chart H.

Given a set of balls (or points) S in chart $H(\theta)$, b_i in S is called *pareto-optimal* if no $b_j \in S$ satisfies $(\alpha_j, \beta_j) > (\alpha_i, \beta_i)$, and $S' \subseteq S$ is called a *pareto-optimal set* of S if all balls in S' are pareto-optimal and no two balls b_i and b_j in S' satisfy $(\alpha_i, \beta_i) = (\alpha_j, \beta_j)$. Note that a maximal pareto-optimal set is always maximum, and pareto-optimal sets correspond to antichains of $P(\theta)$. Moreover, a minimum w-multiple antichain covering of $P(\theta)$ can be computed by the following algorithm which finds (and removes) maximum pareto-optimal sets of B repeatedly.

Algorithm ANTICHAIN-PARTITION

Input: A set of balls $B = \{b_1, b_2, \ldots, b_n\}$.

Output: A minimum antichain partition $(S_1, S_2, \ldots, S_r)$ of B (corresponding to a minimum w-multiple antichain covering of $P(\theta)$)

Step 0: Compute (α_i, β_i) for all $b_i \in S$. Set $S := \{b_i | (\alpha_i, \beta_i) \geqslant (0, 0)\}$ and $r := 1$.

Step 1: Compute a maximal pareto-optimal set S_r in S and remove S_r from S.

Step 2: If $S := \emptyset$, then halt. Otherwise, let $r := r + 1$ and return to Step 1.

Fig. 5 illustrates Algorithm ANTICHAIN-PARTITION. Let $(S_1, S_2, \ldots, S_r)$ be an antichain partition obtained by Algorithm ANTICHAIN-PARTITION. A maximum weighted chain of $P(\theta)$ can be constructed from S_i, $i = 1, \ldots, r$, as follows. Choose a ball b_{i_1} from S_r arbitrarily, and then find a ball b_{i_2} in S_{r-1} such that $(\alpha_{i_2}, \beta_{i_2}) \geqslant (\alpha_{i_1}, \beta_{i_1})$. By the maximality of S_{r-1}, such a b_{i_2} always exits. We then find a ball b_{i_3} in S_{r-2} such that $(\alpha_{i_3}, \beta_{i_3}) \geqslant (\alpha_{i_2}, \beta_{i_2})$. By repeating this argument, we finally have a path $i_0 = 0, i_1, \ldots, i_r$ in $G(\theta)$. This path clearly corresponds to a weighted chain C of $P(\theta)$. Since the weight of chain C is r, it follows from Theorem 3 that $(S_1, S_2, \ldots, S_r)$ corresponding to a minimum w-multiple antichain covering of $P(\theta)$ and C is a maximum weighted chain of $P(\theta)$.

Although we skip the details how to compute a maximal pareto-optimal set S_i in chart $H(\theta)$, due to the space limitation, it is not difficult to see that S_i can be computed (and deleted) in $O(|S_i| \log n)$ time by using a balanced binary search tree such as AVL- and splay trees to store $H(\theta)$. Therefore, we have the following result which improves upon the one in the previous section.

Theorem 4. *We can compute in* $O(n \log n)$ *time a robot schedule that maximizes the number of the balls collected by a robot moving on a given track-line.*

Note that the problem contains the sorting problem, since a robot schedule decides the ordering of balls caught, i.e., sorts the balls caught in the order of t_i. Since any (comparison-based) sorting algorithm needs $\Omega(n \log n)$ time, our algorithm is optimal. We finally remark that our algorithm is based on the fact that each ball crosses at most once, but not the one that balls and robots move on lines, respectively. This means that our method may be applicable in some other situations.

2.2. k-Robot BCP with a given single track-line

This section considers the case when more than 1 robot can be used. Let us first consider the problem (iii), i.e., minimizing the number of robots to catch all the balls. We show that the problem can be solved in $O(n \log n)$ time by using the technique shown in the previous section.

Note that the problem (iii) can be reduced to compute a minimum chain covering of the poset $P(\theta)$. We solve the problem by applying the algorithm similar to ANTICHAIN-PARTITION to the chart $H^*(\theta)$ defined below.

Let $t' = \max_i t_i$, $d' = \min_{i,j:d_i \neq d_j} |d_i - d_j|$, and $v' = v + \delta$ where $0 < \delta < d'/t'$. Let $H'(\theta)$ be a two-dimensional chart with t- and d-axes. $H(\theta)$ has $n+1$ points (t_i, d_i), $i = 0, 1, \ldots, n$, associated with the region R'_i defined by $d \leqslant v'(t - t_i) + d_i$ and $d \geqslant -v'(t - t_i) + d_i$. In other words H' is obtained from H by enlarging each region R_i by a small δ. By this modification, no point (t_i, d_i) is on the boundary of R_j, if $(t_i, d_i) \neq (t_j, d_j)$, and since δ is small, H' still has a good property of H, i.e., a point (t, d) is contained in R_i if and only if a robot starting at state (t_i, d_i) can reach the state (t, d).

Let $\alpha_i^* = -t_i + d_i/v'$ and $\beta_i^* = t_i + d_i/v'$. Let $H^*(\theta)$ denote the chart of the $n+1$ points (α_i^*, β_i^*), $i = 0, 1, \ldots, n$. Similarly to $H(\theta)$, the points (α_i^*, β_i^*) $(i = 1, 2, \ldots, n)$ are identified with balls b_i. We can see that H^* is a dual chart of H in the following sense. Let b_i and b_j be balls such that $(t_i, d_i) \neq (t_j, d_j)$. Then (α_i, β_i) and (α_j, β_j) are comparable (i.e., $(\alpha_i, \beta_i) \leqslant (\alpha_j, \beta_j)$ or $(\alpha_i, \beta_i) \geqslant (\alpha_j, \beta_j)$) if and only if (α_i^*, β_i^*) and (α_j^*, β_j^*) are incomparable.

Since the problem (iii) is to compute a minimum chain covering of the poset $P(\theta)$ (corresponding to $H(\theta)$), the problem (iii) can be solved by computing a minimum antichain covering of the poset $P^*(\theta)$ corresponding to $H^*(\theta)$. Thus, we can use the algorithm similar to ANTICHAIN-PARTITION for the problem. The difference of the algorithms are the following 2 points: Step 0 initializes S as $S = \{b_i \mid \alpha_i^* \leqslant 0, \beta_i^* \geqslant 0\}$ and Step 1 computes the set of all pareto-optimal points instead of a maximal pareto-optimal set. The second difference follows from the one between a minimum antichain covering and a minimum w-multiple antichain covering. Similarly to Theorem 4, we have the following theorem.

Theorem 5. *We can compute in* $O(n \log n)$ *time a robot schedule minimizing the number of the robots on a given single track-line that collect all balls.*

We finally consider the problem (ii) when $k \geqslant 2$. When k is not enough to collect all balls, we can formulate the problem as a minimum cost network flow problem [1,6] of the following network $N = (V', A')$ associated with the capacity, cost and demand functions. The network N can be constructed from $G(\theta) = (V, A)$ as follows

$$V' = \{1^+, \ldots, n^+\} \cup \{1^-, \ldots, n^-\} \cup \{0, n+1\},$$

$$A' = \{(i^+, j^-) \mid (i, j) \in A\} \cup \{(i^-, i^+) \mid i = 1, \ldots, n\} \cup \{(0, i^-), (i^+, n+1) \mid i = 1, \ldots, n\},$$

$$demand(v) = -k \quad \text{if } v = 0, \quad k \text{ if } v = n+1 \text{ and } 0 \text{ otherwise},$$

$$cap(e) = 1 \quad \text{for all } e \in A',$$

$$cost(e) = -1 \quad \text{if } e = (i^+, i^-) \text{ and } 0 \text{ otherwise}.$$

Note that this N does not contain a negative cycle, since otherwise the original $G(\theta)$ contains a directed cycle, which contradicts that $G(\theta)$ is a DAG. Since the demand function *demand* is integer, one can compute a minimum cost integral flow. Although the details are omitted due to the space limitation, we can show that integral feasible flows correspond to k-robot schedules, and moreover, such minimum cost flows correspond to optimal k-robot schedules. Minimum cost flow problem has been extensively studied [1,6]. Here we make use of *primal-dual algorithm with least-cost augmenting paths* to the network N. Since $|V'| = O(n)$, $|A'| = O(n^2)$ and $k \leqslant n$, the problem can be solved in $O(n^3)$ time.

Theorem 6. *We can compute in* $O(n^3)$ *time a k-robot schedule that maximizes the number of the balls collected by k robots moving on a given track-line.*

Before concluding this section, we note that, differently from Theorems 4 and 5, Theorem 6 does not use the property (or structure) of $G(\theta)$, and hence more efficient algorithms may be possible.

3. BCPs with given multiple track-lines

In this section, we address the case that $k(\geqslant 2)$ robots move on their own track-line given.

Let θ_j be an angle for the track-line ℓ_j which robot r_j moves on. We first study the problem (i) when $k = 2$.

Theorem 7. *It can be checked in* $O(n^2)$ *time if all balls can be collected by two robots moving on two given track-lines. Furthermore, if so, such a robot schedule can be computed in* $O(n^2)$ *time.*

Proof. We reduce the problem to the problem 2-SAT whose size is $O(n^2)$. Since 2-SAT is solvable in linear time [7], our problem is solvable in $O(n^2)$ time.

We define a Boolean variable u_i for each ball b_i. Here $u_i = 1$ means that b_i is caught by robot r_1 and $u_i = 0$ means that b_i is caught by robot r_2. A 2-CNF ψ contains the following 4 types of clauses.

1. $\bar{u}_i$, if $G(\theta_1)$ contains no arc $(0, i)$.
2. $(\bar{u}_i \vee \bar{u}_{i'})$, if $G(\theta_1)$ contains neither arc (i, i') nor arc (i', i).
3. u_i, if $G(\theta_2)$ contains no arc $(0, i)$.
4. $(u_i \vee u_{i'})$, if $G(\theta_2)$ contains neither arc (i, i') nor arc (i', i).

It is easy to see that the size of ψ is $O(n^2)$. Note that 3 and 4 are dual of 1 and 2, respectively. By the definitions of $G(\theta_1)$ and $G(\theta_2)$, if all balls can be collected by two robots, ψ is satisfiable. On the other hand, if ψ is satisfiable, we shall show below that all balls can be collected. Let u^* be an assignment such that $\psi(u^*) = 1$, and let $S_1 = \{i \mid u_i^* = 1\}$ and $S_2 = \{i \mid u_i^* = 0\}$. For any pair of i and i' in S_1, $G(\theta_1)$ contains arc (i, i') or (i', i). It follows from the transitivity of $G(\theta_1)$ that it must contain a directed path through all the nodes in S_1. This implies that S_1 can be caught by robot r_1. Similarly, S_2 can be caught by robot r_2. Since $S_1 \cup S_2 = \{1, 2, \ldots, n\}$, two robots can collect all balls. This completes the proof. □

Unfortunately, the problem becomes intractable if we generalize the problem slightly.

Theorem 8. (i) *It is NP-hard to maximize the number of the balls collected by* 2 *robots with* 2 *given track-lines.*

(ii) *It is NP-complete to decide if all balls can be collected by k robots with given track-lines, if* $k \geqslant 3$.

Proof. We only show the proof of Theorem 8(i), since (ii) can also be proved by the similar way.

Since the decision version of this BCP obviously belongs to NP, we concentrate the reduction. Here, we reduce a wellknown NP-complete problem MAX 2SAT [8] to this BCP, which proves this theorem. MAX 2SAT is described as follows:

Problem MAX 2SAT (decision problem)

Input: A 2CNF Boolean Formula ϕ, i.e., a set U of variables and a conjunction ϕ of disjunctive clauses of at most 2 literals, where a literal is a variable or a negated variable in U, and a positive integer p.

Question: Is there an assignment of variables in which the number of satisfied clauses is at least p?

Let $U = \{u_1, u_2, \ldots, u_m\}$ denote a set of variables, where m is the number of variables. A MAX 2SAT instance is defined as $\phi = \bigwedge_j^n c_j = \bigwedge_j^n (l_j^{(1)} \vee l_j^{(2)})$, where $c_j = l_j^{(1)} \vee l_j^{(2)}$, $j = 1, 2, \ldots, n$, are clauses, and literals $l_j^{(1)}$ and $l_j^{(2)}$ are a variable or a negated variable. First, we construct a new CNF ϕ' from a given Max 2SAT instance ϕ. For clause $c_j = l_j^{(1)} \vee l_j^{(2)}$, we define two new clauses as follows:

$$c_j^{(1)} = l_j^{(1)} \vee r_j,$$

$$c_j^{(2)} = l_j^{(2)} \vee \bar{r}_j.$$

By these new clauses $c_j^{(1)}$ and $c_j^{(2)}$ for $j = 1, 2, \ldots, n$, the new MAX 2SAT instance is defined as

$$\phi' = \left(\bigwedge_j c_j^{(1)} \right) \wedge \left(\bigwedge_j c_j^{(2)} \right).$$

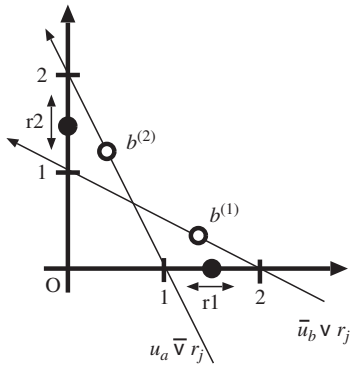

Fig. 6. Balls corresponding to original clause $c_j = (\bar{u}_b \vee u_a)$ (clauses $\bar{u}_b \vee r_j$ and $u_a \vee \bar{r}_j$).

Note that the answer of the original MAX 2SAT instance (ϕ, p) is yes if and only if at least $p + n$ clauses of ϕ' are satisfiable; we can see the answer of the original instance by solving the new instance $(\phi', p + n)$. Therefore, we consider the reduction from the new MAX 2SAT instance ϕ', instead of the original MAX 2SAT instance ϕ.

Now we suppose that in 2-robot BCP the given lines ℓ_1 and ℓ_2 are x-axis (i.e., the track-angle equals to 0) and y-axis (i.e., the track-angle equals to $\pi/2$), respectively (Fig. 6), and that the max speeds of the robots are sufficiently large. Then, suppose balls $b_j^{(1)}$ and $b_j^{(2)}$ corresponding to clauses $c_j^{(1)} = (l_j^{(1)} \vee r_j)$ and $c_j^{(2)} = (l_j^{(2)} \vee \bar{r}_j)$, respectively (i.e., in total $2n$ balls). The balls cross x-axis (robot 1's track-line) and y-axis (robot 2's track-line) as follows: (see Fig. 6)

- For clause $c^{(1)} = (l_j^{(1)} \vee r_j)$: if $l_j^{(1)} = u_i$ (resp., $l_j^{(1)} = \bar{u}_i$), then ball $b_j^{(1)}$ crosses x-axis at $(1, 0)$ (resp., $(2,0)$) at time i and y-axis at $(0, 1)$ at time $n + j$.
- For clause $c^{(2)} = (l_j^{(2)} \vee \bar{r}_j)$: if $l_j^{(2)} = u_i$ (resp., $l_j^{(2)} = \bar{u}_i$), then ball $b_j^{(2)}$ crosses x-axis at $(1, 0)$ (resp., $(2,0)$) at time i and y-axis at $(0, 2)$ at time $n + j$.

We then show that $p + n$ clauses in ϕ' are satisfiable if and only if robots can collect $p + n$ balls in this BCP; i.e., the number of satisfiable clauses is equal to the number of collectable balls. First, we show the if-part. Robot 1 can be either at $(1, 0)$ or at $(2, 0)$ at time i. We decide an assignment of variable set U according to the position of the robots. That is, if robot 1 is located at $(1, 0)$ (resp., $(2, 0)$) at time i, then let $u_i = 1$ (resp., 0). Similarly, we assign $r_j = 1$ (resp., 0) if robot 2 is at $(0, 1)$ (resp., $(0, 2)$). In this assignment, clauses corresponding to the balls caught by robots are satisfied. Therefore, this assignment gives a solution of the new MAX 2SAT instance ϕ', in which the number of satisfied clauses is equal to the number of collected balls, which proves the if-part. Next, we show the only-if part. According to a solution of MAX 2SAT instance ϕ' (i.e., an assignment of u_i for $i = 1, \ldots, m$ and r_j for $j = 1, \ldots, n$), we schedule the location of robot 1 (resp., 2) at time i (resp., $n + j$) according to the assignment of u_i (resp., r_j). Since the speeds of the robots are sufficiently large, they can move as the schedule. The number of balls caught by the robots in the schedule is equal to the number of the satisfied clauses. These complete the proof. □

The next corollary follows from Theorem 8(ii).

Corollary 2. *It is NP-hard to minimize the number of the robots with given track-lines that collect all balls.*

4. BCPs with arbitrary track-lines

In Sections 2 and 3, we consider the BCPs under the assumption that track-lines ℓ_j are given in advance. In this section, we consider the case in which a track-line of each robot can be chosen as a part of the robot scheduling.

4.1. Selecting track-lines

This section shows that the number of the track-lines that have to be considered for the BCPs is polynomial, although there exist infinitely many lines through the origin.

Let θ denote the angle of a line ℓ through the origin. For any ordered pair of i and j in $V(=\{0, 1, \ldots, n\})$, let

$$I_{ij} = \{\theta \in [0, \pi) \mid G(\theta) \text{ contains arc } (i, j)\}.$$

Moreover, let $I_{ij} = \bigcup_p I_{ij}^p$, with interval I_{ij}^p in $[0, \pi)$. Note that I_{ij}^p might not be closed.

Lemma 2. *The number of the intervals I_{ij}^p representing I_{ij} for $i, j \in V$ is* $O(1)$, *and the interval representation of I_{ij} can be computed in* $O(1)$ *time.*

The proof is omitted, since it is rather complicated (see Appendix A.2). The lemma immediately implies the following lemma, since we have n^2 ordered pairs (i, j).

Lemma 3. *There exists* $O(n^2)$ *different directed graphs $G(\theta)$, $\theta \in [0, \pi)$.*

One might see that $O(n^2)$ is overestimated, since $G(\theta)$ contains many transitive arcs, in general. However, we can construct a problem instance having $\Omega(n^2)$ directed graphs $G(\theta)$. By combining Lemma 3 with the results in the previous sections, we have the following results.

Theorem 9. *Assume that a track-line of each robot can be decided as a part of the robot scheduling. Then*

(i) *We can compute in* $O(n^3 \log n)$ *time a robot schedule that maximizes the number of the balls collected by a robot.*
(ii) *We can compute in* $O(n^3 \log n)$ *time a robot schedule minimizing the number of the robots on a single track-line that collect all balls.*
(iii) *We can compute in* $O(n^5)$ *time a k-robot schedule that maximizes the number of the balls collected by k robots moving on a single track-line.*
(iv) *It can be checked in* $O(n^6)$ *time if all balls can be collected by two robots moving on two track-lines. Furthermore, if so, such a robot schedule can be computed in* $O(n^6)$ *time.*

Theorem 9(i)–(iv) follow from Theorems 4–7, respectively.

4.2. Collecting all balls by a single robot

Theorem 9(i) implies that we can decide whether all balls can be collected by a single robot in $O(n^3 \log n)$ time. This section shows that a faster algorithm is possible for the problem.

For the angle θ of a line ℓ, let $E(\theta) = \{(i, j) \mid i < j, G(\theta) \text{ contains arc } (i, j) \text{ or } (j, i)\}$.

Lemma 4. *All balls can be collected by a robot on a line ℓ if and only if $|E(\theta)| = n(n + 1)/2$.*

This lemma implies that it is not necessary to check if $G(\theta)$ has a directed path from 0 whose length is n, but is sufficient to check the size of $E(\theta)$.

Here we only present an outline of the algorithm, due to the space limitation. Recall that I_{ij} can be represented by the union of $O(1)$ intervals. We first compute such representations of all I_{ij}, which requires $O(n^2)$ time. Let $\mathcal{I}$ denotes the family of all such intervals $I = \langle L, R\rangle$. Here $\langle$ is either [or (, and $\rangle$ is either] or). Then we sort all the left and right elements L and R in $\mathcal{I}$, which is possible in $O(n^2 \log n)$ time. Let $J_1, J_2, \ldots, J_q$ be such an ordering. We finally check if $|E(\theta)| = n(n + 1)/2$ for all $\theta = 0, J_1 + \varepsilon, J_2 + \varepsilon, \ldots, J_q + \varepsilon$ in this order, where ε is sufficiently small positive. ε is used for open intervals. It is not difficult to see that these angles are sufficient to check. Since this can be done in $O(n^2)$ time, we have an $O(n^2 \log n)$ time algorithm.

Theorem 10. *It can be checked in* $O(n^2 \log n)$ *time if there exists a track-line ℓ such that a robot can collect all balls by moving on ℓ. Furthermore, if so, such a robot schedule (including ℓ) can be computed in* $O(n^2 \log n)$ *time.*

4.3. The NP-hardness cases

Similarly to Theorem 8, we have the negative results even if the track-lines can be chosen.

Theorem 11. *Even if a track-line of each robot can be chosen as a part of the robot scheduling, the following statements holds.*

(i) *It is* NP*-hard to maximize the number of the balls collected by* 2 *robots with* 2 *track-lines.*
(ii) *It is* NP*-complete to decide if all balls can be collected by k robots with track-lines, if* $k \geqslant 3$.

5. Conclusion and discussions

We have studied the problem of collecting balls moving in the Euclidean plane by robots moving on straight lines through the origin. From the viewpoint of computational complexity, we have investigated the following 3 problems in various situations: (i) deciding if k robots can collect all n balls; (ii) maximizing the number of the balls collected by k robots; (iii) minimizing the number of the robots to collect all n balls. The results are summarized in Table 1 in Section 1.

There are many possible extensions of the problems. For example, we can relax the track shapes of robots and/or balls. Unfortunately, it is shown that the problems become intractable, if robots and/or balls move on general tracks.

Acknowledgements

The authors would like to thank the anonymous referee for their helpful comments which improved the presentation of this paper.

Appendix A

A.1. Details of algorithm ANTICHAIN-PARTITION

In Section 2.1.2, we propose an $O(n \log n)$ algorithm to maximize the number of the balls collected by 1 robot on a given track line. Algorithm ANTICHAIN-PARTITION computes a maximal pareto-optimal set S_i in S in Step 1. In this section, we show the details of finding a maximal pareto-optimal set S_i, which can be done in $O(|S_i| \log n)$ by utilizing a balanced binary search tree.

First, we introduce procedure PARETO(p, S), to find a point $b_{i^*} \in S$ such that y_{i^*} is maximum among the points in S whose α value is larger than p. If such b_{i^*} does not exist, return $\emptyset$. (Fig. 7)

By procedure PARETO, the details of Step 1 of algorithm ANTICHAIN-PARTITION is described as follows:

Step 1 of Algorithm ANTICHAIN-PARTITION.
Input: A set of points S,
Output: A maximal pareto-optimal set S_r in S, and $S := S \backslash S_r$,
Step 1-0: Set $S_r := \emptyset$ and $p := 0$.
Step 1-1: Call PARETO(p, S), and let its output be b_{i^*}. If $b_{i^*} = \emptyset$, then let $S := S \backslash S_r$, return S and halt.
Otherwise, let $S_r := S_r \cup \{b_{i^*}\}$ and then return to Step 1-1.

Clearly Step 1 finds a maximal pareto-optimal set in S. In the rest of this section, we consider an efficient implementation of procedure PARETO by adding some devise to standard balanced binary search tree data structures. The ideas of the devise are as follows:ce:displayed-quotece:simple-paraEach node of the binary tree corresponds to an element of S, and keeps three values; the corresponding element's (α, β) and $\tilde{\beta}$, the maximum value of β among node b_i and its successor nodes. The binary search tree is constructed by the normal rule according to only key value α. That is, α value of each node is not smaller than its left child's α and is smaller than its right child's α. Fig. 8 shows an example of the binary search tree, where three values $((a, b), c)$ at each nodes corresponds to α, β and $\tilde{\beta}$, respectively. Now, we consider

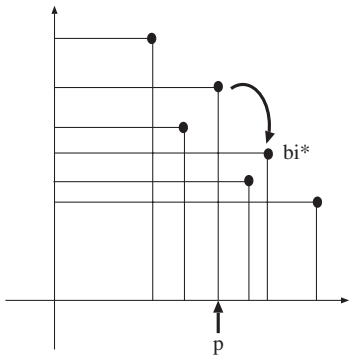

Fig. 7. Procedure PARETO.

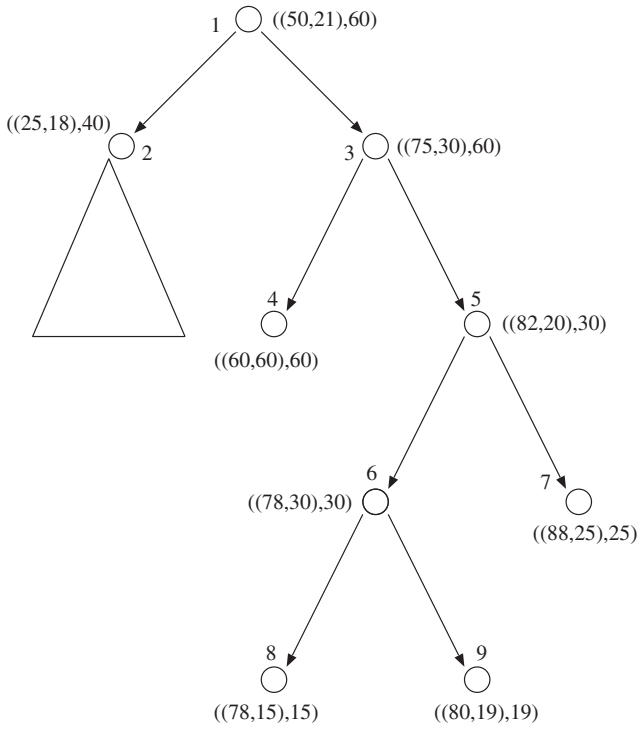

Fig. 8. An example of the binary search tree.

how to execute procedure PARETO(p, S). For simplicity, in the explanation, we assume that all β values are different, without loss of generality.

Procedure PARETO(p, S)

Input: A balanced binary tree S, defined as above, p: a value.

Output: $b_{i^*} \in S$ such that y_{i^*} is maximum among the points in S whose α value is larger than p, if exists. Otherwise, return $\emptyset$.

Step 0: Traverse the binary tree by moving to the right child until finding the node b_i such that $\alpha_i \geqslant p$. (We call this node *base*.) Then, set $c^* := \tilde{\beta}_i$ and goto Step 2-1. If such a node does not exist, then return $\emptyset$ and halt.

Step 1: If $\alpha_i \geqslant p$, then goto Step 2-1, and if $\alpha_i < p$ then goto Step 2-2.
Step 2-1: ($\alpha_i \geqslant p$)] If $\beta = c^*$ holds, then return b_i and halt. Otherwise, check if b_i has only a right child, or the left child $b_{i'}$ satisfies $\tilde{\beta}_{i'} < c^*$. If yes, goto Step 3. Otherwise, move to $b_{i'}$ (i.e., set $i := i'$) and goto Step 1.
Step 2-2: ($\alpha_i < p$)] Check if b_i is a leaf node, or the right child $b_{i'}$ satisfies $\tilde{\beta}_{i'} < c^*$. If yes, goto Step 4. Otherwise, move to $b_{i'}$ (i.e., set $i := i'$) and goto Step 1.
Step 3: Move to the right child $b_{i''}$, and traverse the subtree by moving to the node satisfying $\tilde{\beta}_{\hat{i}} = c^*$, until $\beta_{\hat{i}} = c^*$ holds. Return $b_{\hat{i}}$ as b_{i^*} and halt.
Step 4: If b_i is not a leaf node, call PARETO(p, S'), where S' is the subtree from b_i, and let the output be $b_{i^{**}}$. (If b_i is not a leaf node, set $b_{i^{**}} := \emptyset$.) Backtrack to the base node as we check the right children of all the intermediate nodes; find the maximum $\tilde{\beta}$ value c^{**}. If $\beta_{i^{**}}$ is larger than c^{**}, then output $b_{i^{**}}$. Otherwise, move to c^{**} node and goto Step 3.

For example, PARETO(70, S) in Fig. 8 outputs $b_6 = (78, 30)$ as follows: First, node 3 is defined as the base node and $c^* := 60$ (Step 0). Move to node 4 (Step 2-1, Step 1, Step 2-2). Since node 4 is a leaf node, go to Step 4. Backtrack to node 3 and its right child 5, then node set $c^{**} := 30$ and traverse node $5 \to 6$ ($\beta_6 = c^{**}$).

All the steps of PARETO(p, S) can be done in linear time of the height of the tree. If all the basic operations, such as INSERT, BALANCE, DELETE and FIND, can be implemented in the same computational time as standard balanced binary search trees, the height of the tree is also kept in $O(\log |S|)$; i.e., PARETO(p, S) can be done in $O(\log |S|)$. In fact, all the operations can be done in $O(\log |S|)$. The reasons are as follows: Our binary search tree is different from standard balanced binary search trees in the point that only our tree needs to keep the values of $\tilde{\beta}$. However, since only the comparison between children's $\tilde{\beta}$ and its own β is needed for the update of $\tilde{\beta}$, all the extra computational steps are absorbed in the order of original computational steps of INSERT and so on. That is, procedure PARETO(p, S) can be executed in $O(|S_i| \log n)$.

Therefore, we have the following lemma, which leads to Theorem 4.

Lemma 5. *In Step* 1 *of algorithm* ANTICHAIN-PARTION, S_i *can be computed in* $O(|S_i| \log n)$.

A.2. The proof of Lemma 2

In this subsection, we show the proof of Lemma 2, which is omitted in Section 4.1.

Lemma 2. *The number of the intervals* I_{ij}^p *representing* I_{ij} *for* $i, j \in V$ *is* $O(1)$, *and the interval representation of* I_{ij} *can be computed in* $O(1)$ *time.*

Proof. We consider the condition that a robot who caught ball b_i can catch ball b_j after catching b_i. We consider the case that $\theta \in (-\pi/2, \pi/2)$ instead of $[0, \pi)$, since they are essentially same and it simplifies the proof. Let $i = 1$ and $j = 1$ in this proof without loss of generality. Ball b_1 and b_2 start at their initial points $p_1^{(0)} = (x_1^{(0)}, y_1^{(0)})$ and $p_2^{(0)} = (x_2^{(0)}, y_2^{(0)})$ with velocities $\mathbf{v}_1 = (v_1 \cos\varphi_1, v_1 \sin\varphi_1)$ and $\mathbf{v}_2 = (v_2 \cos\varphi_2, v_2 \sin\varphi_2)$, respectively. Balls b_1 and b_2 cross the track line, so

$$\tan\varphi_1 \neq \tan\theta \quad \text{and} \quad \tan\varphi_2 \neq \tan\theta.$$

The points at which b_1 and b_2 cross the track line are

$$p_1(\theta) = \left(\frac{y_1^{(0)} - x_1^{(0)} \tan\varphi_1}{\tan\theta - \tan\varphi_1}, \frac{y_1^{(0)} - x_1^{(0)} \tan\varphi_1}{\tan\theta - \tan\varphi_1} \tan\theta \right),$$

and

$$p_2(\theta) = \left(\frac{y_2^{(0)} - x_2^{(0)} \tan\varphi_2}{\tan\theta - \tan\varphi_2}, \frac{y_2^{(0)} - x_2^{(0)} \tan\varphi_2}{\tan\theta - \tan\varphi_2} \tan\theta \right),$$

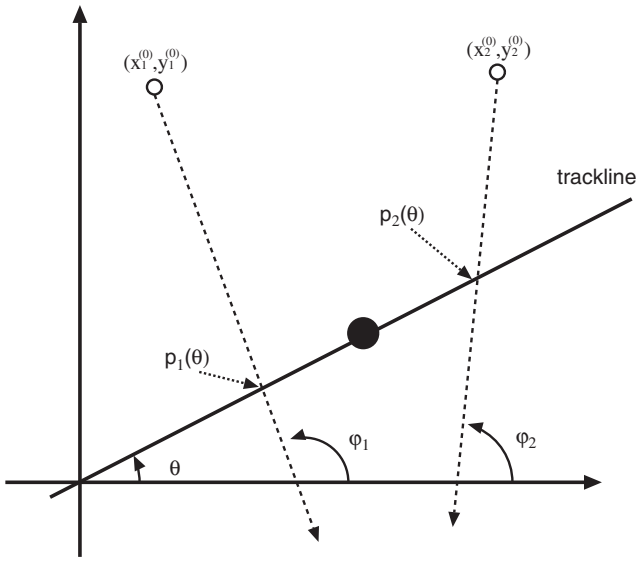

Fig. 9. A robot, balls b_1 and b_2, and trackline.

respectively (see Fig. 9). The time when b_1 touches the track line is

$$\begin{aligned} t_1(\theta) &= \overline{p_1^{(0)} p_1(\theta)}/v_1 \\ &= \frac{1}{v_1}\sqrt{\left(\frac{y_1^{(0)} - x_1^{(0)}\tan\varphi_1}{\tan\theta - \tan\varphi_1} - x_1^{(0)}\right)^2 + \left(\frac{y_1^{(0)} - x_1^{(0)}\tan\varphi_1}{\tan\theta - \tan\varphi_1}\tan\theta - y_1^{(0)}\right)^2} \\ &= \frac{|y_1^{(0)} - x_1^{(0)}\tan\theta|}{v_1|\cos\varphi_1||\tan\theta - \tan\varphi_1|}, \end{aligned}$$

and the time when b_2 touches the track line is

$$\begin{aligned} t_2(\theta) &= \overline{p_2^{(0)} p_2(\theta)}/v_2 \\ &= \frac{1}{v_2}\sqrt{\left(\frac{y_2^{(0)} - x_2^{(0)}\tan\varphi_2}{\tan\theta - \tan\varphi_2} - x_2^{(0)}\right)^2 + \left(\frac{y_2^{(0)} - x_2^{(0)}\tan\varphi_2}{\tan\theta - \tan\varphi_2}\tan\theta - y_2^{(0)}\right)^2} \\ &= \frac{|y_2^{(0)} - x_2^{(0)}\tan\theta|}{v_2|\cos\varphi_2||\tan\theta - \tan\varphi_2|}, \end{aligned}$$

where the distance between points p_a and p_b is denoted by $\overline{p_a p_b}$. Here, the condition that the robot catching b_1 can catch b_2 is

$$\overline{p_1(\theta)p_2(\theta)} \leqslant v(t_2(\theta) - t_1(\theta)), \tag{1}$$

where

$$\overline{p_1(\theta)p_2(\theta)} = \left|\frac{y_1^{(0)} - x_1^{(0)}\tan\varphi_1}{\tan\theta - \tan\varphi_1} - \frac{y_2^{(0)} - x_2^{(0)}\tan\varphi_2}{\tan\theta - \tan\varphi_2}\right|\sqrt{1 + \tan^2\theta}.$$

By inequality (1), we obtain two inequalities, $t_2(\theta) \geqslant t_1(\theta)$ and $\overline{p_1(\theta)p_2(\theta)}^2 \leqslant v^2(t_2(\theta) - t_1(\theta))^2$. The former

$$\frac{|y_2^{(0)} - x_2^{(0)} \tan\theta|}{v_2|\cos\varphi_2||\tan\theta - \tan\varphi_2|} \geqslant \frac{|y_1^{(0)} - x_1^{(0)} \tan\theta|}{v_1|\cos\varphi_1||\tan\theta - \tan\varphi_1|} \tag{2}$$

yields four quadratic inequalities of $\tan\theta$. The conditions can divide interval I_{12} into at most 9 parts, since $\tan\theta$ is a monotonically increasing function. The latter is transformed into

$$\begin{aligned}&((y_1^{(0)} - x_1^{(0)} \tan\varphi_1)(\tan\theta - \tan\varphi_2) - (y_2^{(0)} - x_2^{(0)} \tan\varphi_2)(\tan\theta - \tan\varphi_1))^2(1 + \tan^2\theta)\\ &\quad\leqslant \frac{1}{v^2}\left(\frac{|y_2^{(0)} - x_2^{(0)} \tan\theta||\tan\theta - \tan\varphi_1|}{v_2|\cos\varphi_2|} - \frac{|y_1^{(0)} - x_1^{(0)} \tan\theta||\tan\theta - \tan\varphi_2|}{v_1|\cos\varphi_1|}\right)^2,\end{aligned} \tag{3}$$

which also yields four quadratic inequalities of $\tan\theta$, so the interval can be divided into at most 21 parts. Hence, the condition that a robot who caught ball b_1 can catch ball b_2 after catching b_1 divides interval I_{12} into O(1) intervals. Moreover, these conditions can be computed in O(1), since the degrees of polynomial inequalities (2) and (3) are both at most 4; the formulae of computing the roots are known. □

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] Y. Asahiro, T. Horiyama, K. Makino, H. Ono, T. Sakuma, M. Yamashita, How to Collect Balls Moving in the Euclidean Plane, Electronic Notes in Theoretical Computer Science, vol. 91, 2004, pp. 229–245.

[3] M. Atallarh, S. Kosaraju, Efficient solutions to some transportation problems with applications to minimizing robot arm travel, SIAM J. Comput. 17 (1988) 849–869.

[4] S. Browne, U. Yechiali, Scheduling deteriorating jobs on a single processor, Oper. Res. 38 (1990) 495–498.

[6] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, Combinatorial Optimization, Wiley, New York, 1998.

[7] S. Evan, A. Itai, A. Shamir, On the complexity of timetabling and multicommodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.

[8] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, New York, 1979.

[9] M. Hammar, B.J. Nilsson, Approximation results for kinetic variants of TSP, 26th International Colloquium, Automata, Languages and Programming, ICALP'99, Lecture Notes in Computer Science, vol. 1644, 1999, pp. 392–401.

[10] C.S. Helvig, G. Robins, A. Zelikovsky, Moving target TSP and related problems, Sixth Annual European Symposium, Algorithms—ESA '98, Lecture Notes in Computer Science, vol. 1461, 1998, pp. 453–464.

[11] Y. Karuno, H. Nagamochi, T. Ibaraki, Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks, Networks 39 (4) (2002) 203–209.

[12] D.E. Knuth, Fundamental Algorithms, Addison-Wesley, Reading, MA, 1997.

[13] L. Lovász, M.D. Plummer, Matching Theory, Annals of Discrete Mathematics, vol. 29, North-Holland, Amsterdam, 1986.

[14] C.L. Monma, C.N. Potts, On the complexity of scheduling with batch setup times, Oper. Res. 37 (1989) 798–814.

[15] G. Mosheiov, V-shaped policies for scheduling deteriorating jobs, Oper. Res. 39 (6) (1991) 979–991.

[16] H.N. Psaraftis, M.M. Solomon, T.L. Magnanti, T. Kim, Routing and scheduling on a shoreline with release times, Management Sci. 36 (2) (1990) 212–223.

[18] J.N. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, Networks 22 (1992) 263–282.

[19] S.T. Webster, K.R. Baker, Scheduling groups of jobs on a single machine, Oper. Res. 43 (1995) 692–703.