Better Upper Bounds for Searching on a Line with Byzantine Robots

Xiaoming Sun[1,2], Yuan Sun[1,2], and Jialin Zhang[1,2]

[1] CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{sunxiaoming,sunyuan2016,zhangjialin}@ict.ac.cn

Abstract. Searching on a line with Byzantine robots was first posed by Jurek Czyzowicz et al. in [13]: Suppose there are n robots searching on an infinite line to find a target which is unknown to the robots. At the beginning all robots stay at the origin and then they can start to search with maximum speed 1. Unfortunately, f of them are *Byzantine fault*, which means that they may ignore the target when passing it or lie that they find the target. Therefore, the target is found if at least $f+1$ robots claim that they find the target at the same location. The aim is to design a parallel algorithm to minimize the competitive ratio $S(n, f)$, the ratio between the time of finding the target and the distance from origin to the target in the worst case by n robots among which f are Byzantine fault.

In this paper, our main contribution is a new algorithm framework for solving the Byzantine robot searching problem with (n, f) sufficiently large. Under this framework, we design two specific algorithms to improve the previous upper bounds in [13] when $f/n \in (0.358, 0.382) \cup (0.413, 0.5)$. Besides, we also improve the upper bound of $S(n, f)$ for some small (n, f). Specifically, we improve the upper bound of $S(6, 2)$ from 4 to 3.682, and the upper bound of $S(3, 1)$ from 9 to 8.53.

Keywords: Searching on a line, Mobile robots, Parallel search, Competitive ratio, Byzantine fault.

1 Introduction

1.1 Problem Description

In this paper, we consider the following robot searching problem from [13]. Suppose there are n robots searching on a one-dimensional axis in parallel, and their aim is to find a target placed somewhere on the line unknown to these robots. At the beginning all robots are at the origin of the axis. The maximum speed of each robot is 1 per unit time, both on moving either along the positive direction (which can be seen as moving right) or negative direction (which can be seen as moving left). During the searching, any robot can change its direction at any

position without loss of time. The state of a position is detected by a robot only when this robot has passed there. Robots are assumed to have full knowledge of the detected searching states at any time. The mission is to find the target as fast as possible.

If all robots are functioning normally, this problem is quite trivial. However, when we introduce *Byzantine fault* into the robot system, the problem becomes much complicated. In fact, when we build a communication network to control the actions of all robots, it is natural to consider the fault tolerance of the network. We say a robot is *Byzantine fault* or a *Byzantine robot*, if during the searching process it may ignore the target or lie that it finds the target at its position. In order to deal with Byzantine faults, we allow that robots can *communicate* and *vote*. All robots can communicate with each other in wireless mode with any distance. Communication does not cost time. When a robot claims it finds the target, it also set up a vote for this claim in the communication channel. Before the vote ends any robot passing the controversial position should vote on supporting or opposing this claim. If we know that there are f Byzantine robots in the robot system, then the vote ends immediately when at least $f+1$ robots holding the same opinion on one side. More precisely, when there are at least $f+1$ robots claiming that the target is found at the same position, we can finish the searching process and believe that we find the target indeed. Similarly, when there are at least $f+1$ robots showing that a finding claim is false, we can mark all the robots which support the claim as Byzantine robots. We allow that there are more than one votes in the communication channel simultaneously.

Under the Byzantine robot model, we need some evaluation functions to judge the efficiency of a protocol among all robots. One obvious way to evaluate a searching algorithm is to consider the ratio between time and the distance from the origin in the worst case. In the next subsection, we will define such functions formally.

1.2 Evaluation Functions

For the problem that n robots among which are f byzantine fault ones search on a line, we define two kinds of functions, $S(n, f)$ and $T(\frac{f}{n})$, to represent the optimal search time among all algorithms for the searching problem. This notations is similar to the ones in [13].

First, the concept *searching time* need be clarified:

Definition 1 (Searching Time). *Given two integers n, f, a real number x and a protocol (or an algorithm) $\mathcal{A}$, we define $S_x^{\mathcal{A}}(n, f)$ as the searching time of the situation that under the algorithm $\mathcal{A}$, $(n-f)$ normal robots with f Byzantine fault robots together find the target whose coordinate is x on the axis.*

Then we can evaluate the efficiency of an algorithm by the following *competitive ratio* function:

Definition 2 (Competitive Ratio). *Define*

$$S^{\mathcal{A}}(n,f) = \sup_{|x|>0}\left\{\frac{S_x^{\mathcal{A}}(n,f)}{|x|}\right\}$$

as the competitive ratio of the algorithm $\mathcal{A}$ with the parameter pair (n,f). In other words, this function represents the ratio of the searching time and distance in the worst case.

Remark 1. W.o.l.g, we may need to assume that the target can not be placed to close to the origin. For example, we can suppose that the distance between the target and the origin is at least 0.01.

For the whole problem, first let $\mathscr{A}$ be the set of all algorithms for this problem. Then we use

$$S(n,f) = \inf_{\mathcal{A}\in\mathscr{A}}\{S^{\mathcal{A}}(n,f)\}$$

to be the competitive ratio of the Byzantine robot searching problem with a robot system with fault ratio β.

When both n and f are both sufficiently large, we turn to consider the relationship between $\frac{f}{n}$ (we say $\frac{f}{n}$ is the *fault ratio* of a robot system with f Byzantine robots and $(n-f)$ normal robots) and the competitive ratio, which induces the *asymptotic competitive ratio* function:

Definition 3 (Asymptotic Competitive Ratio). *For $\beta \in (0, \frac{1}{2})$, define*

$$T(\beta) = \overline{\lim_{n\to\infty}} S(n, \beta n)$$

as the asymptotic competitive ratio of the Byzantine robot searching problem with a robot system with fault ratio β.

Remark 2. For competitive ratio, the parameter (n,f) we are interested in satisfies $2f+1 \le n \le 4f+1$, because if $n \le 2f$ then we can not judge any position where f robots claim while others give different opinion, and if $n \ge 4f+2$ then we just let two groups with $\frac{n}{2}$ robots search each side [13].

For asymptotic competitive ratio, we only consider the fault ratio β satisfying $\frac{1}{4} < \beta < \frac{1}{2}$ for the same reason. Furthermore, for convenience the status of a position is totally checked if at least $2\beta n$ robots have passed there, instead of $2\beta n + 1$.

1.3 Our Results

The main contribution of this work is that, for sufficiently large n and f we provide an algorithm framework, ByzantineSearch(Base, μ), to give upper bounds of asymptotic competitive ratio. Here, Base is a searching algorithm which can only deal with the robot systems with fault ratio no more than μ. The details of this algorithm framework are in Algorithm 1, Section 3. As a brief

description, the main process of ByzantineSearch is that if the fault ratio of the current robot system is no more than μ we just call Base, otherwise we call a recursion subroutine repetitively to reduce the fault ratio. In the recursion subroutine, when some robots claim that they find the target, we do not let others come to check immediately, but let them move along their original routes for a while then turn to check. The deferred time on checking depends on an extra parameter. This operation is the key to the trade-off among all bad cases. We merge all extra parameters in each times of recursive program into a parameter sequence. For a given (Base, μ) we can give an optimal assignment to the parameter sequence to get the best upper bound of asymptotic competitive ratio under the algorithm ByzantineSearch(Base, μ).

As a result, we achieve the upper bounds of asymptotic competitive ratio in Theorem 1 and Theorem 2 by ByzantineSearch(Base, μ) with two different (Base, μ) pairs:

Theorem 1. *Define a real number sequence* $\{\beta_n\}_{n\geq 1} : \beta_0 = \frac{5}{14}, \beta_n = \frac{1+\beta_{n-1}}{4-2\beta_{n-1}} (n \geq 1)$ *and a function sequence* $\{f_n(x)\}_{n\geq 1} : f_1(x) = x+1, f_n(x) = (x+1)(1 - \frac{1}{f_{n-1}(x)})(n \geq 2)$. *Then if the fault ratio* β *satisfies* $\beta_{n-1} < \beta \leq \beta_n$ *for some* $n \geq 1$, *we have* $T(\beta) \leq 3 + 2x^*$, *where* $x^* \in (0,3)$ *is the largest real root of the equation* $f_n(x) = \frac{x+1}{x}$.

Theorem 2. *Define a real number sequence* $\{\beta_n\}_{n\geq 1} : \beta_0 = \frac{13}{34}, \beta_n = \frac{1+\beta_{n-1}}{4-2\beta_{n-1}} (n \geq 1)$ *and a function sequence* $\{f_n(x)\}_{n\geq 1} : f_1(x) = x+1, f_n(x) = (x+1)(1 - \frac{1}{f_{n-1}(x)})(n \geq 2)$. *Then if the fault ratio* β *satisfies* $\beta \leq \beta_0$, *we have* $T(\beta) \leq 3.682$; *otherwise if the fault ratio* β *satisfies* $\beta_{n-1} < \beta \leq \beta_n$ *for some* $n \geq 1$, *we have* $T(\beta) \leq 3 + 2x^*$, *where* $x^* \in (0,3)$ *is the largest real root of the equation* $f_n(x) = \frac{x+1}{x-0.341}$.

β	$(\frac{5}{14}, \frac{13}{34}]$	$(\frac{13}{34}, \frac{19}{46}]$	$(\frac{19}{46}, \frac{47}{110}]$	$(\frac{47}{110}, \frac{65}{146}]$
Previous Upper Bounds	4	5	6	7
Our Upper Bounds	3.682	5	5.682	6.236
Achieved by	Theorem 2	Theorem 1	Theorem 2	Theorem 1
β	$(\frac{65}{146}, \frac{157}{346}]$	$(\frac{157}{346}, \frac{211}{454}]$	$(\frac{211}{454}, \frac{503}{1070}]$	$(\frac{503}{1070}, \frac{665}{1394}]$
Previous Upper Bounds	8	9	9	9
Our Upper Bounds	6.747	7	7.369	7.494
Achieved by	Theorem 2	Theorem 1	Theorem 2	Theorem 1
β	$(\frac{665}{1394}, \frac{1573}{3274}]$	$(\frac{1573}{3274}, \frac{2059}{4246}]$	$(\frac{2059}{4246}, \frac{4847}{9950}]$	...
Previous Upper Bounds	9	9	9	9
Our Upper Bounds	7.762	7.828	8.026	...
Achieved by	Theorem 2	Theorem 1	Theorem 2	...

Table 1. Upper bounds of $T(\beta)$ when $\frac{5}{14} < \beta < \frac{1}{2}$. (The previous upper bounds are from [13])

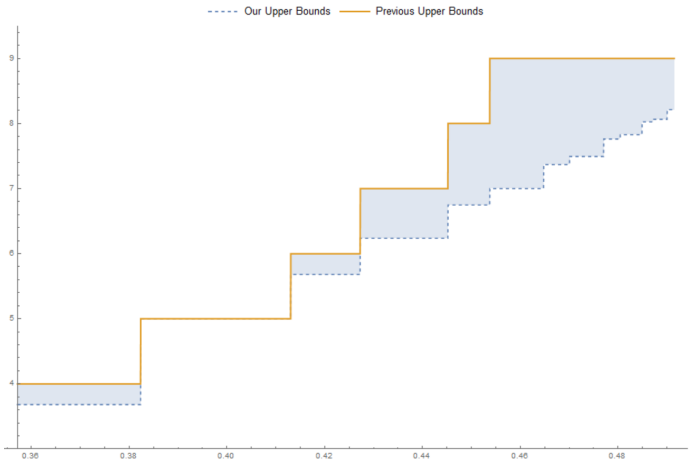

Fig. 1. The comparison between the efficiency of previous and current results. X axis represents the ratio parameter β, and Y axis represents the upper bounds of $S(\beta)$. The straight line is the previous results while the dashed line is the results achieved in this paper.

The final upper bounds are achieved by combining Theorem 1 and Theorem 2. Table 1 gives an intuitive version of the above theorems, and compares our results with the previous ones. It can be seen that when $\beta \in (\frac{5}{14}, \frac{13}{34}] \cup (\frac{19}{46}, \frac{1}{2})$, Theorem 1 and Theorem 2 make significant improvement. Moreover, Figure 1 presents a straightforward visual feeling of the improvement.

For some small (n, f), we also make achievement on the upper bounds of $S(n, f)$. First, a corollary of Theorem 2 directly gives a better upper bound of $S(6, 2)$:

Corollary 1. $S(6, 2) \leq 3.682$.

Next, for the case where $(n, f) = (3, 1)$, we get the following theorem by a totally different algorithm, which is based on an algorithm from [11]:

Theorem 3. $S(3, 1) \leq 8.653$.

Table 2 compares these results with the previous ones.

(n, f)	Previous Upper Bounds	Our Upper Bounds	Lower Bounds
$(3, 1)$	9	8.653	5.23
$(6, 2)$	4	3.682	3

Table 2. Upper bounds of $S(n, f)$ for some small n and f. (The previous upper bounds are from [13], and lower bounds are from [13,19])

1.4 Related Works

Searching on a line is an interesting combinatorial problem with a long history. The original version of this problem appears in [3], where a robot needs to search on an infinite line to find an unknown target. It shows that the competitive ratio is exact 9. Since then, there are a series of follow-up discussions [2,3,4,5,6].

If we change the searching region, there are many interesting extension of this problem. Some researchers expand the detectable region to higher dimensions [16,18,22]. To get more complicated environment, researchers also introduce obstacles [1], or just put all agents in a large network [7,14]. Another variant is the so-called *cow-path problem* [17], where a robot need to search on several rays shared with a same origin. This problem is solved by [15], and [17] gives an optimal randomized solution to it.

Searching by a group of detectors also leads out a number of works. [8] considers the simplest form of the linear group search in which the process ends when the target is reached by the last robot visiting it and shows that increasing the number of robots does not help and the optimal competitive ratio is still 9. [10,14] think of the cost and restrictions of communications and information exchanges between the group members.

In group searching, it is natural to consider the safety of the whole system. The concept *Byzantine Fault Tolerance* has been widely studied in distributed computing [21]. [20] notices that it is important to keep the whole system safe even if some of the agents are broken down and send wrong messages to others, which induces the *Byzantine general problem*. On group searching with faulty members, there are also many impressive works [9,10]. Recently, researchers discuss the problem that the robots search on a line by group together while some of them are crash or Byzantine fault [11,13,19]. These is the main model discussed in this paper. See also [12] as a survey.

1.5 Organization

In Section 2, we will denote some necessary notations to describe our algorithm better. In Section 3, we will give an algorithm framework for the asymptotic competitive ratio. In Section 4, we provide two specific base algorithms which can be used as inputs for the framework and prove main theorems. In Section 5, we turn to discuss the competitive ratio for small n and f. In Section 6, we conclude our work and give some open problems.

2 Notations

In order to describe our algorithms clearly, we provide some new definitions.

First, we give several notations on robot systems.

Definition 4 (Robot Status Representation (RSR)). *For a robot r, its robot status representation (RSR) is a tuple (pos_r, d_r), where $pos_r \in R$ is its coordinate and $d_r \in \{-1, 0, 1\}$ is its current moving direction. Here $d_r = 1$ (-1)*

means r will move along the right (left) direction after receiving move order, and $d_r = 0$ means r will stop at its position after receiving move order.

Definition 5 (Group Searching System (GSS)). *At any time of the searching progress, the status of all working robots can be described as a group searching system (GSS) $\mathcal{P} = (n, f, R, S)$, where n is the number of robots, f is the number of Byzantine fault robots, R is the set of all robots and S is the set of all RSRs. Note that we do not know which f robots are Byzantine fault. In a GSS, if all robots in a set A is at a same position, we say that the position of A is the position of robots in A.*

Definition 6 (Symmetric Two-group Searching System (STSS)). *A GSS $\mathcal{P}$ is a symmetric two-group searching system (STSS) if all robots in $\mathcal{P}$ can be divided into two groups A and B such that:*

- *Robots in the same group have the same RSR tuple;*
- *for any two robots $r_1 \in A$ and $r_2 \in B$, $pos_{r_1} = -pos_{r_2}$ and $d_{r_1} = -d_{r_2}$.*

We write an STSS $\mathcal{P}$ as $\mathcal{P} = (n, f, A, B, (pos, d))$, where (pos, d) is the RSR of robots in the set A. In particular, a STSS $\mathcal{P}$ is an initial STSS, if all robots are at the origin, and half of robots have directions -1 while others have directions 1. An initial STSS can also be seen as a status of a robot system at the beginning of searching.

Next, we define some basic orders on a robot system, which can be used as basic components of our algorithms.

- Move($\mathcal{P}$, R, t): for the GSS P, keep robots in the robot set R moving along their directions for time t with speed 1. In other words, this function will cost t units of time and for any robot $r \in R$ after this function its position will be $pos_r := pos_r + td_r$. Robots in R will check their position to find the target while moving.
- MoveAll($\mathcal{P}$, t): for the GSS P, keep all robots moving along their directions for time t with maximum speed. All robots will check their position to find the target while moving.
- TurnAround($\mathcal{P}$, R): for the GSS P, change directions of robots in the robot set R. i.e., set $d_r := -d_r$ for all $r \in R$.
- TurnAroundAll(P): for the GSS P, change directions of all robots.
- Remove($\mathcal{P}$, R): for the GSS P, remove all robots in R from $\mathcal{P}$.
- Join($\mathcal{P}$, R_1, R_2, k): for the GSS $\mathcal{P}$, arbitrarily choose k robots from the robot set R_1, and add them to the robot set R_2. This operation only works when k is no larger than $|R_1|$ and the positions of robots in both R_1 and R_2 are all the same.
- Select($\mathcal{P}$, R_0, k): for the GSS $\mathcal{P}$ and a robot set R_0, this operation will randomly choose k robots in R_0, and return a robot set containing these k robots. This operation only work when $k \leq |R_0|$ and $R_0 \subseteq \mathcal{P}.R$.

Also, here are some specific variables closely related to the searching process:

- *TargetClaim*: a Boolean variable which is False initially, but turns to be True immediately after some robots claim the finding of the target, while the number of robots with opposite opinion is no more than f. This variable will turn to be False again when all robots reach a consensus that the claim on *ClaimPos* is false, and turn to be True for the next effective claim, and so on.
- *ClaimPos*: a real number representing the position where some robots claim they find the target which change the value of $TargetClaim$ from False to True. (This variable is meaningless when *TargetClaim* is False.)
- *ClaimSet*: The set of robots which claim they find the target (This variable is meaningless when *TargetClaim* is False.)
- *TargetVeri*: a Boolean variable which is False initially, but turns to be True immediately after the target is actually found. (i.e., at least $f+1$ robots claim they find the target at the same place.) After it turns to be True, its value will not be changed.
- *TargetPos*: a real number representing the position of the target (This variable is meaningless when *TargetVeri* is False.)

Remark 3. Here are some extra supplementary explanation for the operations and variables in Section 2.2.

- It is easy to see that the time cost of a searching process is the sum of time parameters in Move and MoveAll functions.
- When we use Move order some robots, the others stay at their positions.
- Positions on the line can be verified even when *TargetClaim* is true, but the value of $TargetPos$ won't be changed when a new claim occurs but $TargetClaim$ is already True.
- If more than one positions are claimed to be the location of target which makes $TargetClaim$ from False to True, randomly choose one position as the value of $ClaimPos$.
- If during a Move process a position is verified to be the real position of the target, the Move function will be ended immediately and values of the two variables $TargetPos$ and $TargetVeri$ will be updated.

3 An Algorithm Framework for Byzantine Robot Searching Problem

In this section we provide an algorithm framework, ByzantineSearch, for the Byzantine robot searching problem. The detail is stated in Algorithm 1.

Algorithm 1 ByzantineSearch(Base, μ): A searching algorithm framework

1: $\mathcal{P}$:=Initial STSS
2: $i := 0$, m :=Times($\mathcal{P}.f/\mathcal{P}.n, \mu$)
3: **while** $(\mathcal{P}.f/\mathcal{P}.n > \mu) \wedge (TargetVeri = \text{False})$ **do**
4: $\quad i := i + 1$

5: RECUR($\mathcal{P}$, $\lambda(i,m)$)
6: **if** $TargetVeri =$ True **then**
7: **return** $TargetPos$
8: **else**
9: BASE($\mathcal{P}$)

This framework has two input parameters, BASE and μ. BASE is a *partial* searching algorithm, and μ is the corresponding fault ratio limit. More precisely, BASE can deal with the Byzantine searching problem with an STSS only when the fault ratio is no more than μ. In Section 4, we will provide two different (BASE, μ) to Algorithm 4 and prove Theorem 1 and Theorem 2.

At the beginning of BYZANTINESEARCH, the robot system turns to be an initial STSS $\mathcal{P}$. If the fault ratio is no more than μ, we can just call BASE algorithm. Otherwise, we provide a searching subroutine called RECUR. Each time after we call RECUR, either we find the target or the fault ratio of STSS $\mathcal{P}$ is reduced. Thus we can call BASE algorithm when the fault ratio of STSS $\mathcal{P}$ is reduced to be no more than μ.

In BYZANTINESEARCH, the procedure RECUR is the core component, which can reduce the fault ratio and keep the robot system still an STSS. The process details of RECUR are stated in Algorithm 2. The RECUR procedure has two inputs, an STSS $\mathcal{P}$ and a real number $\lambda \geq 1$. The basic idea is that first let $\mathcal{P}$ search until some robots claim, then all robots turn around and move again. When all robots meet at the origin, two groups swap some members. After this subroutine either the target is found or $\mathcal{P}$ is still an STSS but with lower fault ratio. However, if each time when some robots claim others move to check immediately, the time cost will be too large if the target is near $ClaimPos$. To balance this bad case, the key point of RECUR is deferred verification. When some robots claim that they find the target, robots in the other group will not turn around to check it immediately but move along their directions for a while. The moving length depends on λ, the second input. In BYZANTINESEARCH, the i-th time we call RECUR the value of the second input λ is $\lambda(i,m)$. $\{\lambda(i,m)\}(1 \leq i \leq m)$ is a real parameter sequence, whose value is not related to the correctness of BYZANTINESEARCH, but can affect the (asymptotic) competitive ratio. For a given (BASE, μ), we can give an optimal assignment to the parameter sequence to get the best upper bound of asymptotic competitive ratio under the algorithm BYZANTINESEARCH(BASE, μ), by solving a convex optimization. In the proof of Theorem 4 we will give an example on this assertion.

To give an intuitive impression, the process of Algorithm 2 is also showed in Figure 2.

Algorithm 2 RECUR($\mathcal{P}$, λ): A group searching subroutine with an STSS $\mathcal{P}$ and a real number $\lambda \geq 1$ to reduce the value of $\mathcal{P}.f/\mathcal{P}.n$

1: **do**
2: MOVEALL($\mathcal{P}$, 0.01) ▷ Here 0.01 can be a arbitrarily small number.
3: **until** $TargetClaim =$ True
4: MOVEALL($\mathcal{P}$, $(\lambda - 1) \cdot |ClaimPos|$)

5: TurnAroundAll($\mathcal{P}$)
6: MoveAll($\mathcal{P}, \lambda \cdot |ClaimPos|$) ▷ After this order, all robots are at the origin.
7: $m := |ClaimSet|$, $k := \min\{n - 2f, m/2\}$
8: **if** $ClaimSet \subseteq \mathcal{P}.A$ **then**
9: Join($\mathcal{P}$, $\mathcal{P}.B$, $\mathcal{P}.A$, k)
10: **else**
11: Join($\mathcal{P}$, $\mathcal{P}.A$, $\mathcal{P}.B$, k)
▷ After this order $\mathcal{P}$ turns to be a general GSS.
12: R' :=Select($\mathcal{P}$, $ClaimSet$, $2k$)
13: MoveAll($\mathcal{P}$, $\lambda \cdot |ClaimPos|$)
14: Remove($\mathcal{P}$, R') ▷ After this order $\mathcal{P}$ turns to be an $STSS$ again.

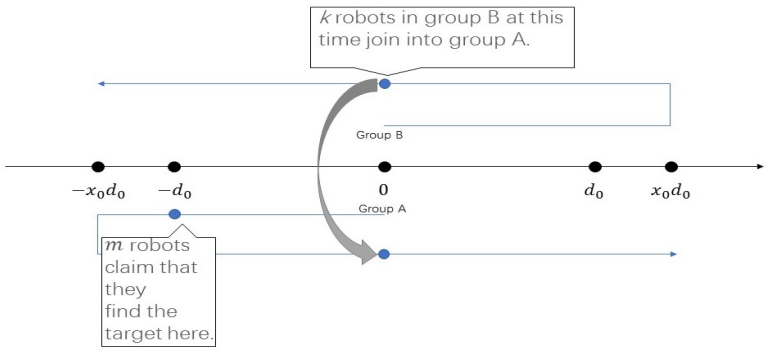

Fig. 2. Process of Algorithm 2

Now we show some basic properties of Recur. It is easy to verify the next two facts:

Fact 1 *For any STSS $\mathcal{P}$, it will still be an STSS after* Recur($\mathcal{P}$, λ).

Fact 2 *For any STSS $\mathcal{P}$, all points in the interval $[-\lambda \cdot |ClaimPos|, \lambda \cdot |ClaimPos|]$ will totally be checked after* Recur($\mathcal{P}$, λ).

Next, Lemma 1 shows the ability of Recur on reducing the fault ratio of $\mathcal{P}$:

Lemma 1. *For a given STSS $\mathcal{P}$, let $\beta = \frac{\mathcal{P}.f}{\mathcal{P}.n}$. Then either* Recur($\mathcal{P}$, λ) *actually finds the target, or it can reduce β to a number no greater than $\frac{4\beta-1}{2\beta+1}$. (Notice that $\beta < \frac{1}{2}$, so we always have $\frac{4\beta-1}{2\beta+1} < \beta$.)*

Proof. For an STSS $\mathcal{P} = (n, f, A, B, (pos, d))$ with fault ration $\beta = \frac{f}{n}$, suppose there are m robots claiming the finding after line 3 in Algorithm 2. Note that we have $m \geq \frac{n}{2} - f$, otherwise the claiming will be verified as a false statement

immediately by all robots at $ClaimPos$. Thus after $\textsc{Recur}(\mathcal{P}, \lambda)$, if the finding claim is false, the fault ratio will turn to be

$$\frac{f-m}{n-m} \leq \frac{1-(\frac{1}{2\beta}-1)}{\frac{1}{\beta}-(\frac{1}{2\beta}-1)} = \frac{4\beta-1}{2\beta+1},$$

or

$$\frac{f-(2n-4f)}{n-(2n-4f)} = \frac{5f-2n}{4f-n} \leq \frac{5\beta-2}{4\beta-1} \leq \frac{4\beta-1}{2\beta+1}.$$

By Lemma 1, if we have a Base algorithm which can deal with the Byzantine robots searching problem with fault ratio no more than μ, then for a given initial STSS $\mathcal{P}$ with fault ratio β, we can easily calculate the maximum possible times that we call Recur to reduce the fault ratio of $\mathcal{P}$ no more than μ, which is the function $\textsc{Times}(\beta, \mu)$. The details of the function Times are stated in Algorithm 3.

Algorithm 3 TIMES(β, μ): The maximum possible recursion times in ByzantineSearch

```
if β ≤ μ then
    return 0
else
    return Times((4β − 1)/(2β + 1), μ) + 1
```

4 Two Different Base Algorithms

In this section, we give two different (Base, μ) pairs to prove Theorem 1 and Theorem 2 respectively.

4.1 A Base Algorithm with $\mu \leq \frac{5}{14}$

For an initial STSS $\mathcal{P}$ with fault ratio no more than 5/14, [11] gives an algorithm in Theorem 13. Here we modify this algorithm as a base algorithm called LeftRight. We give details of LeftRight in Algorithm 4. The main idea is that, after checking $ClaimPos$ if $TargetClaim$ is false, we still keep two groups of sufficient number robots move along different directions respectively after remove Byzantine robots. (That is why we call this algorithm "LeftRight".) Here we omit the proof of the correctness of Algorithm 4, since it is similar to the proof of Theorem 13 in [13].

Algorithm 4 $\textsc{LeftRight}(\mathcal{P})$: A base algorithm for an STSS $\mathcal{P}$ satisfying $\frac{\mathcal{P}.f}{\mathcal{P}.n} \leq \frac{5}{14}$

```
do
    MoveAll(P, 0.01)    ▷ Here 0.01 can be an arbitrary small positive real number.
until TargetClaim = True    ▷ W.l.o.g, suppose ClaimSet ⊆ P.A.
```

4: R_1 :=SELECT($\mathcal{P}$, $\mathcal{P}.B$, $(3/5)|\mathcal{P}.B|$)
5: R_2 :=SELECT($\mathcal{P}$, $ClaimSet$, $(2/5)|ClaimSet|$)
6: R_3 :=SELECT($\mathcal{P}$, $\mathcal{P}.A \setminus ClaimSet$, $(2/5)|\mathcal{P}.A \setminus ClaimSet|$)
7: TURNAROUND($\mathcal{P}$, $R_1 \cup R_2 \cup R_3$) ▷ After this order $\mathcal{P}$ turns to be a general GSS.
8: MOVE($\mathcal{P}$, $R_1 \cup R_2 \cup R_3$, $2|ClaimPos|$)
9: **if** $TargetVeri =$ False **then**
10: REMOVE($\mathcal{P}$, $ClaimSet$)
11: **do**
12: MOVEALL($\mathcal{P}$, 0.01)
13: **until** $TargetVeri =$ True
14: **return** $TargetPos$

Combining BYZANTINESEARCH and LEFTRIGHT, we have the following result leading to Theorem 1:

Theorem 4. BYZANTINESEARCH(LEFTRIGHT, 5/14) *with a specific parameter sequence* $\{\lambda(i,m)\}$ *can achieve the upper bounds of asymptotic competitive ratio in Theorem 1.*

Proof. First, it is easy to verify that the sequence $\{\beta_n\}$ in Theorem 1 is monotonically increasing and $\lim_{m\to\infty}\beta_m = \frac{1}{2}$. So for a given β with $\frac{5}{14} < \beta < \frac{1}{2}$, there exists an integer $m \geq 1$ such that $\beta_m < \beta \leq \beta_{m+1}$.

Second, suppose $\mathcal{P}$ is an initial STSS with $\frac{\mathcal{P}.f}{\mathcal{P}.n} = \beta$, by Lemma 1 and the definition of $\{\beta_n\}$ we know that the **while** loop in BYZANTINESEARCH(LEFTRIGHT, 5/14) will repeat at most m times. Denote that in the $i-$th $(1 \leq i \leq m)$ repetition of the **while** loop the value of the parameter $\lambda(i,m)$ is λ_i, and let $\lambda_0 = 1$. Our aim is to choose specific $\{\lambda_i\}$ to reduce the asymptotic competitive ratio.

There are two cases on finding the target. First, suppose in the $i-$th $(1 \leq i \leq m)$ repetition of the **while** loop we actually find the target. Then the worst case will be that the target is placed at $\pm(\lambda_{i-1}ClaimPos + \epsilon)$, where $\epsilon > 0$ is an arbitrary small number. Second, the target is found in the base algorithm LEFTRIGHT. Then the worst case will be that the target is placed at $\pm(\lambda_m ClaimPos + \epsilon)$, where $\epsilon > 0$ is an arbitrary small number. After calculating the searching time of all bad cases, we can express the asymptotic competitive ratio of BYZANTINESEARCH(LEFTRIGHT, 5/14) as

$$\max\{3+2\lambda_1, 3+2\lambda_2+\frac{2}{\lambda_1}, 3+2\lambda_3+\frac{2}{\lambda_2}+\frac{2}{\lambda_2\lambda_1}, ...,$$

$$3+2\lambda_m+\frac{2}{\lambda_{m-1}}+\frac{2}{\lambda_{m-1}\lambda_{m-2}}+...+\frac{2}{\lambda_{m-1}\lambda_{m-2}...\lambda_1},$$

$$5+\frac{2}{\lambda_m}+\frac{2}{\lambda_m\lambda_{m-1}}+...+\frac{2}{\lambda_m\lambda_{m-1}...\lambda_1}\}$$

By analysis in the convex optimization, we know that if $(\lambda_1, \lambda_2, ..., \lambda_m)$ satisfies:

$$3 + 2\lambda_1 = 3 + 2\lambda_2 + \frac{2}{\lambda_1} = ... = 5 + \frac{2}{\lambda_m} + \frac{2}{\lambda_m\lambda_{m-1}} + ... + \frac{2}{\lambda_m\lambda_{m-1}...\lambda_1}, \quad \lambda_1 \geq 1, \lambda_2 \geq 1, ..., \lambda_m \geq 1 \tag{1}$$

then $3 + 2\lambda_1$ will be the upper bound of the asymptotic competitive ratio, and this assignment to the parameter sequence $\{\lambda(i, m)\}$ is optimal for Byzantine-Search(LeftRight, 5/14).

To solve (1) we let $x = \lambda_1$ and use the first two equations $3 + 2\lambda_1 = 3 + 2\lambda_2 + \frac{2}{\lambda_1}$. Then we get that $\lambda_2 = \frac{x^2-1}{x}$. Similarly we can express $\lambda_i (i > 2)$ as $\lambda_i = (1 + x)(1 - \frac{1}{\lambda_{i-1}})$ by the equations $3 + 2\lambda_1 = 3 + 2\lambda_i + \frac{2}{\lambda_i} + \frac{2}{\lambda_i\lambda_{i-1}} + ... + \frac{2}{\lambda_i\lambda_{i-1}...\lambda_1}$, $3 \leq i \leq m$. Combining these results to the equation $3 + 2\lambda_1 = 5 + \frac{2}{\lambda_m} + \frac{2}{\lambda_m\lambda_{m-1}} + ... + \frac{2}{\lambda_m\lambda_{m-1}...\lambda_1}$, and then we have:

$$\begin{aligned}(1 - \frac{1}{\lambda_m})(3 + 2x) =& (5 + \frac{2}{\lambda_m} + \frac{2}{\lambda_m\lambda_{m-1}} + ... + \frac{2}{\lambda_m\lambda_{m-1}...\lambda_1}) \\ & - \frac{1}{\lambda_m}(3 + 2\lambda_m + \frac{2}{\lambda_{m-1}} + \frac{2}{\lambda_{m-1}\lambda_{m-2}} + ... + \frac{2}{\lambda_{m-1}\lambda_{m-2}...\lambda_1})\end{aligned} \tag{2}$$

Finally the equation turns to be:

$$\lambda_m = \frac{x + 1}{x} \tag{3}$$

Since all λ_i $(1 \leq i \leq m)$ can be expressed as a rational function of x, (3) is a polynomial equation about x. The last thing is to verify that there exists a real root x satisfying that $\lambda_i \geq 1$, $\forall 1 \leq i \leq m$. Lemma 2 and Lemma 3 show that, the largest real root of (3) can meet our requirements.

Lemma 2. *Define a rational function sequence about x: $f_1(x) = x + 1, f_n(x) = (1 + x)(1 - \frac{1}{f_{n-1}(x)})$ $(n \geq 2)$. Let $\{\alpha_n\}$ be the sequence of the largest real root of the equation $f_n(x) = \frac{x+1}{x}$. Then sequence $\{\alpha_n\}$ is monotonically increasing with finite limit and its limit is no greater than 3.*

Proof. Define a rational function sequence $\{g_n(x)\}$ related to $\{f_n(x)\}$: $\forall n \geq 1$, $g_n(x) = f_n(x) - \frac{x+1}{x}$. Then $\{\alpha_n\}$ be the sequence of the biggest real root of the equation $f_n(x) = \frac{x+1}{x} \iff \{\alpha_n\}$ be the sequence of the biggest real root of the equation $g_n(x) = 0$. Note that $f_{n+1}(x) = (1 + x)(1 - \frac{1}{f_n(x)})$, $n \geq 1$, which induces the recursive formula of $g_n(x)$:

$$g_{n+1}(x) = (1 + x)\frac{x^2 g_n(x) - x g_n(x) - 1}{x^2 g_n(x) + x^2 + x}, \ n \geq 1.$$

Now we can prove this lemma by induction. In fact, we will show a much stronger conclusion as the following list:

- $g_n(3) = \frac{2n+6}{3n}$.
- $\forall n \geq 2$, $g_n(x)$ is monotonically increasing and continuous when $x \geq \alpha_n$;
- $\alpha_{n-1} \leq \alpha_n < 3$;

The case of $n = 2$ is trivial. Suppose this proposition holds when $n \leq k$. Now consider $n = k + 1$. First, we have

$$g_{k+1}(3) = 4\frac{6g_k(3) - 1}{9g_k(3) + 12} = 4\frac{6\frac{2k+6}{3k} - 1}{9\frac{2k+6}{3k} + 12} = \frac{2(k+1)+6}{3(k+1)}.$$

Second, calculate the derivation of $g_{k+1}(x)$, we have

$$g'_{k+1}(x) = \frac{1 + 2x^3 g_k(x) + 2x(1 + g_k(x)) + x^2(1 + g_k(x))^2 + x^5 g'_k(x) + x^4(g_k(x) + g_k^2(x) + g'_k(x))}{x^2(1 + x + xg_k(x))^2}.$$

Since when $x \geq \alpha_k$, we have $g_k(x) \geq g_k(\alpha_k) = 0$ and $g'_k(x) > 0$ by induction hypothesis, we have

$$x^2(1 + x + xg_k(x))^2 > 0$$

and

$$1+2x^3 g_k(x)+2x(1+g_k(x))+x^2(1+g_k(x))^2+x^5 g'_k(x)+x^4(g_k(x)+g_k^2(x)+g'_k(x)) > 0,$$

so $g_{k+1}(x)$ is monotonically increasing and continuous when $x \geq \alpha_k$.

Finally, since

$$g_{k+1}(\alpha_k) = (1 + x)\frac{x^2 g_k(\alpha_k) - xg_k(\alpha_k) - 1}{x^2 g_k(\alpha_k) + x^2 + x} = -\frac{1}{x} < 0,$$

there exists exactly one root x_0 of $g_{k+1}(x)$ which lies in the interval $(\alpha_k, 3)$. Because $g_{k+1}(x)$ is monotonically increasing and continuous when $x \geq \alpha_{k+1}$, we have $\alpha_{k+1} = x_0$. Thus $\alpha_k \leq \alpha_{k+1} < 3$.

Combining the above discussions, we finish the induction step of $n = k + 1$.

Lemma 3. *Define a function sequence about x: $f_1(x) = x + 1, f_n(x) = (1 + x)(1 - \frac{1}{f_{n-1}(x)})(n \geq 2)$. Let $\{\alpha_n\}$ be the sequence of the largest real root of the equation $f_n(x) = \frac{x+1}{x}$. Then $\forall n$, if $x \geq \alpha_n$, we get that $f_i(x) > 1$ and $f'_i(x) > 0$, $\forall 1 \leq i \leq n$.*

Proof. We still prove this proposition by induction. When $n = 1$, we have $\alpha_1 = 1$ and $\forall x \geq 1$, $f_1(x) = 1 + x > 1$ and $f'_1(x) = 1 > 0$. Now suppose that this lemma holds for all $1 \leq n \leq k$. Consider about $n = k + 1$. Since $\alpha_{k+1} \geq \alpha_k$ from Lemma 2, by induction hypothesis we just need to prove that $f_{k+1}(x) > 1$ and $f'_{k+1}(x) > 0$ when $x \geq \alpha_{k+1}$. First, we have

$$f'_{k+1}(x) = 1 - \frac{1}{f_k(x)} + (1 + x)(1 + \frac{f'_k(x)}{f_k^2(x)}) > 1 - \frac{1}{1} + \frac{(1 + x)f'_k(x)}{f_k^2(x)} > 0.$$

So if $x \geq \alpha_{k+1}$, $f_{k+1}(x)$ is monotonically increasing. Since α_{k+1} is a root of $f_{k+1}(x) = \frac{x+1}{x}$, we get that

$$f_{k+1}(x) \geq f_{k+1}(\alpha_{k+1}) = \frac{\alpha_{k+1} + 1}{\alpha_{k+1}} > 1,$$

which finish the induction step of $n = k + 1$.

4.2 A Base Algorithm with $\mu \leq \frac{13}{34}$

In this subsection we design a new base algorithm, LeftMiddleRight, which accepts an STSS $\mathcal{P}$ with fault ratio no more than 13/34 as input. The details of this algorithm are stated in in Algorithm 5, which is totally different from LeftRight.

Here we give a brief description for the process. Algorithm 5 can be divided into two parts. The first part of it is from line 1 to line 15, and the **if** component that starts at line 16 is the second part. For a given STSS $\mathcal{P}$, if some robots claim they find the target at the first part, we repartition all robots into 5 groups ($ClaimSet$, A_2, $A_1 \setminus A_2$, B_1, $B \setminus B_1$) and let 4 of them (except $ClaimSet$) move along different routes. At the end of the first part, $ClaimPos$ is checked. If the claim is true, we can finish the searching, otherwise we remove all Byzantine robots and merge remained robots into 3 groups (A_2, B_2, C_2). The position of robots in A_2 and B_2 are symmetric about the position of robots in C. (That is why we call this algorithm "LeftMiddleRight".) Then we let robots in A_2 and B_2 move along their directions until $TargetClaim$ turns to be true again. At this time, robots in C move to $ClaimPos$ to check it.

Algorithm 5 LeftMiddleRight($\mathcal{P}$): A base algorithm for an STSS $\mathcal{P}$ with $\frac{\mathcal{P}.f}{\mathcal{P}.n} \leq \frac{13}{34}$

1: **do**
2: MoveAll($\mathcal{P}$, 0.01)
3: **until** $TargetClaim$ = True ▷ W.l.o.g, suppose $ClaimSet \subseteq \mathcal{P}.A$.
4: $m := |ClaimSet|$, $k := (1/2)n + (2/3)m - (2/3)f$
5: $x_0 := 1.341$, $x_1 := 1.797$, $x_2 := 2.41$
6: $A := \mathcal{P}.A$, $B := \mathcal{P}.B$
7: $A_1 := A \setminus ClaimSet$, B_1 :=Select($\mathcal{P}$, B, k)
8: Move($\mathcal{P}$, $A_1 \cup B$, $x_0|ClaimPos|$) ▷ After this order $\mathcal{P}$ turns to be a general GSS.
9: TurnAround($\mathcal{P}$, B_1)
10: Move($\mathcal{P}$, $A_1 \cup B$, $(x_1 - x_0)|ClaimPos|$)
11: A_2 :=Select($\mathcal{P}$, A_1, $\frac{4}{3}(f - m)$)
12: TurnAround($\mathcal{P}$, A_2)
13: Move($\mathcal{P}$, $A_2 \cup B$, $(x_2 - x_1)|ClaimPos|$)
14: TurnAround($\mathcal{P}$, $B \setminus B_1$)
15: Move($\mathcal{P}$, $A_2 \cup B$, $(x_2 + x_1 - x_0)|ClaimPos|$)
16: **if** $TargetVeri$ = False **then**
17: Remove($\mathcal{P}$, $ClaimSet$)
18: $B_2 := (A_1 \setminus A_2) \cup B_1$, $C := B \setminus B_1$
19: $d := (x_1 - x_0)ClaimPos$ ▷ At this moment, Robots in $B \setminus B_1$ are at d.
20: **do**
21: Move($\mathcal{P}$, $A_2 \cup B_2$, 0.01)
22: **until** $TargetClaim$ = True ▷ W.l.o.g, suppose $ClaimSet \subseteq A_2$.
23: **if** $TargetVeri$ = False **then**
24: Move($\mathcal{P}$, C, $d - ClaimPos$)

25: **if** $TargetVeri =$ False **then**
26: REMOVE($\mathcal{P}$, $ClaimSet$)
27: **do**
28: MOVEALL($\mathcal{P}$, 0.01)
29: **until** $TargetClaim =$ True
30: **return** $ClaimPos$

To give an intuitive impression, the first part of Algorithm 5 is also showed in Figure 3.

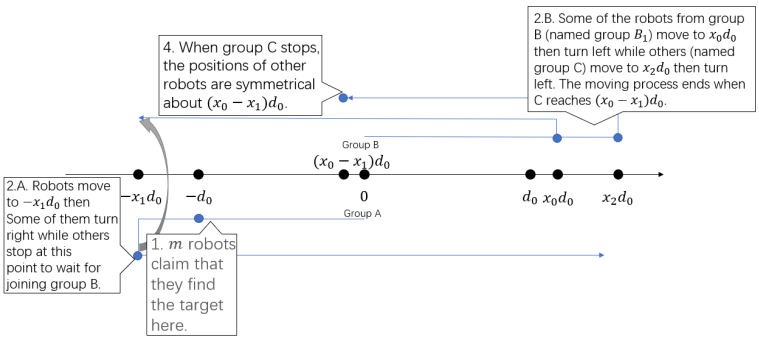

Fig. 3. Process of Algorithm 5

Lemma 4. *Given an STSS $\mathcal{P}$ with fault ratio no more than* 13/34, LEFTMIDDLERIGHT*($\mathcal{P}$) can actually find the target.*

Proof. For any given STSS $\mathcal{P}$ with fault ratio no more than $\frac{13}{34}$, in order to prove that LEFTMIDDLERIGHT($\mathcal{P}$) actually finds the target, let us analyze the algorithm step by step.

In the first part, if the target is truly lying in the interval $[-x_1d_0, -d_0] \cup [d_0, x_0d_0]$, the searching process will stop immediately and the asymptotic competitive ratio will be no more than $1 + 2x_0$ because all the points in the interval $[-x_1d_0, x_0d_0]$ are checked by at least $2\beta n$ robots; If the target is lying in the interval $[x_0d_0, x_2d_0]$, then it will be verified immediately after the robots from group A_2 check its position; Otherwise after the first part the asymptotic competitive ratio of $\mathcal{P}$ turns to be

$$\beta' = \frac{f-m}{n-m} \leq \frac{1 - (\frac{1}{2\beta} - 1)}{\frac{1}{\beta} - (\frac{1}{2\beta} - 1)} \leq \frac{3}{10}.$$

Now let us consider the second part of the algorithm. If $\beta \leq \frac{13}{34}$, then $\beta' \leq \frac{3}{10}$ and at the beginning the group A_2 and B_2 are is symmetrical about $(x_0 - x_1)d_0$ (the position group C stays), and the ratio between the number of robots in A_2,

B_2, C is $2:2:1$. We let the groups A_2 and B_2 keep on moving until $TargetClaim$ turns to be true again. We then let robots in group C move $ClaimPos$. Notice that there will be at least $\frac{3}{5}n' = 2f'$ robots checking $ClaimPos$, so the state of the position can be clearly determined. If the claim is true, then we can finish the search process. If not, then after the REMOVE function $\mathcal{P}.f/\mathcal{P}.n$ will be no greater than $(\frac{3}{10} - \frac{1}{10})/(1 - \frac{1}{10}) = \frac{2}{9} < \frac{1}{4}$, and the distribution of the robots at this time allows us to use an extra MOVE process to finish the searching.

Combining BYZANTINESEARCH and LEFTMIDDLERIGHT, we have the following theorem leading to Theorem 2, whose proof is similar to the one of Theorem 4:

Theorem 5. BYZANTINESEARCH(LEFTMIDDLERIGHT, 13/34) *with a specific parameter sequence* $\{\lambda(i,m)\}$ *can achieve the upper bounds of asymptotic competitive ratio in Theorem 2.*

5 Competitive Ratio $S(n, f)$ for Small n and f

In this section we discuss the upper bounds of competitive ratio $S(n, f)$ for small n and f.

For the case where $n = 6$ and $f = 2$, we design an algorithm similar to Algorithm 5. First, let 3 robots (we mark them as A, B, C) move left while others move right (we mark them as D, E, F). W.l.o.g, suppose A claims that it finds the target at position $-d_0 (d_0 > 0)$. If B or C support this claim, then let D and E turn round immediately to $-d_0$ to check the position while F still moves on. In this situation the competitive ratio will be no greater than 3. Otherwise B and C oppose the claim. Then we let B and C move to $-x_1 d_0$ then turn right, while D and E move to $x_0 d_0$ then turn left and F moves to $x_2 d_0$ then turns left until it stops at the location. $(x_0 - x_1)d_0$. (Here $x_0 = 1.341$, $x_1 = 1.797$, $x_2 = 2.41$.) It is not hard to verify that by this searching process, $S(6, 2) \leq 3.682$, which leads to Corollary 1.

For the case where $n = 3$ and $f = 1$, we will use some results from the problem searching on a line with *crash* robots, where crash means that some robots cannot detect the target. [13] gives an efficient algorithm to deal with this problem. For completeness, we also give the details of the proof of this algorithm.

Lemma 5 (J. Czyzowicz et al. [13]). *There is an algorithm for the case* $(n, f) = (3, 1)$ *that can solve the problem in the version of crash robots with competitive ratio* $(\beta + 1)^{\frac{4}{3}}(\beta - 1)^{-\frac{1}{3}} + 1$ *(*$\beta > 1$ *is a parameter that can be determined by ourselves).*

Proof (Proof of Lemma 5). We provide an algorithm to achieve this competitive ratio. At first we let all robots move right. For robot i $(1 \leq i \leq 3)$, we define its turning-around location sequence as $x_{i,0}, x_{i,1}, x_{i,2}, \ldots$. The first turning-around location of each robot are calculated by the formulas below: $x_{1,0} = 1$, $x_{2,0} =$

$(\frac{\beta+1}{\beta-1})^{\frac{2}{n}}\beta - \beta + 1$, $x_{3,0} = (\frac{\beta+1}{\beta-1})^{\frac{4}{n}}\beta - \beta + 1$. The later turning-around locations of each robot can be calculate by the formulas below: $\forall 1 \leq i \leq 3$, $j \geq 1$, $x_{i,j} = x_{i,j-1} \cdot (\frac{-\beta+1}{\beta-1})$. To give a direct impression, the relationships between time and locations of each robot can be viewed in the left side of Figure 4. This strategy can give the competitive ratio $(\beta+1)^{\frac{4}{3}}(\beta-1)^{-\frac{1}{3}} + 1$ for $S(3,1)$.

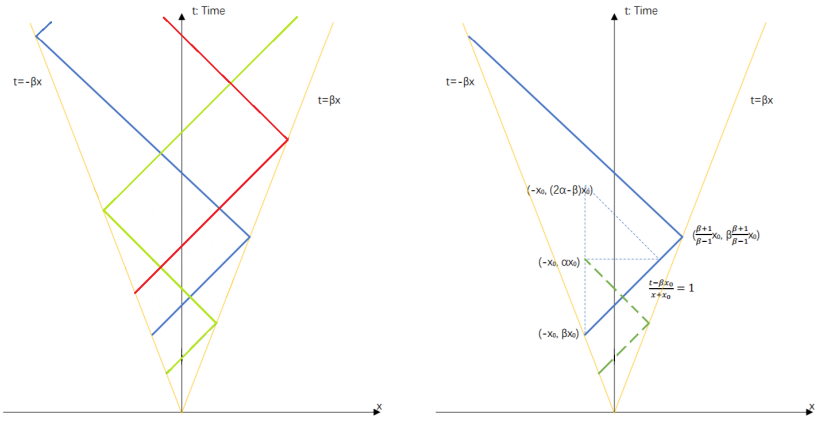

Fig. 4. Left side: The coordinate map between time and distance in the proof of Lemma 5. The three lines represent the traces of each robot. Right side: The coordinate map between time and distance in the proof of Theorem 3. The thick dashed line represents the behavior of the robots which claims that it finds the target near $-x_0$. The straight line represents the behavior of the robot whose turning point is $-x_0$, while the thin dashed line represents its changing after receiving the target detecting message.

We design Algorithm 6 to prove Theorem 3. In the Byzantine fault version, we split the search process into two parts, and express it as follows:

Algorithm 6 ByzantineSearch31($\mathcal{P}$): A searching algorithm with an initial STSS $\mathcal{P}$ where $n = 3$ and $f = 1$

1: In the first part, we run the algorithm described in the proof of Lemma 5, until there is a robot claiming that it finds the target. At the end of this part, w.l.o.g suppose a robot claims that it finds the target at position $-x_0(x > 0)$.
2: In the second part, at the beginning for the other two robots if some of them have not detected the position $-x_0$ and are moving right, let them turn around immediately to move forward to $-x_0$. After checking the claim, if it is false then there remain two normal robots. The final step is letting them moving on opposite directions.

Proof (Proof of Theorem 3). There are many situations need to be discussed in the analysis of Algorithm 6. Here we provide the analysis of the worst situation,

where the target is truly at $-x_0$ which is ignored by the first robot passing there, and the claiming is by the second robot reaching the position $-x_0$. In the first part of Algorithm 6, the search time is αx_0, where $\alpha = (\beta+1)^{\frac{4}{3}}(\beta-1)^{-\frac{1}{3}}+1$ ($\beta > 1$ is a parameter undetermined yet). Notice that there are two robots having detected the position $-x_0$, in which one claims that there is the target while the other holds the different opinion. Now consider the only robot which has not check $-x_0$ yet. We build a coordinate map between the time and coordinate from original axis (horizontal axis represents original coordinate, and vertical axis represents time, see also the right side of Figure 4), we can get that the intersection point of two lines $t = \beta x$ and $\frac{t-\beta x_0}{x+x_0} = 1$ is $(\frac{\beta+1}{\beta-1}x_0, \beta\frac{\beta+1}{\beta-1}x_0)$, so if there was no alert it would move to the point $\beta\frac{\beta+1}{\beta-1}x_0$ then change the direction. So if $\beta\frac{\beta+1}{\beta-1} > \alpha$ we can save time. The total time of the third robot from the origin to $-x_0$ in its orbit is $(2\alpha-\beta)x_0$. Let $\beta = 2.023$, then the competitive ratio is $(2\alpha-\beta) = 2(\beta+1)^{\frac{4}{3}}(\beta-1)^{-\frac{1}{3}}+2-\beta = 8.653$, which also satisfies $\beta\frac{\beta+1}{\beta-1} > \alpha$. Thus the proof of this case is finished.

6 Conclusions and Open Problems

In this paper we investigate the Byzantine robots search problem where n robots search on a line for an unknown target while f of them are Byzantine fault. For sufficiently large (n, f), we provide a new algorithm framework, and design two specific algorithm under this framework. With these algorithms, We improve the upper bounds of asymptotic competitive ratio $T(\beta)$ significantly with the fault ratio $\beta \in (\frac{5}{14}, \frac{13}{34}) \cup (\frac{19}{46}, \frac{1}{2})$ compared with the results in [13]. Besides, we also improve upper bounds of competitive ratio $S(n, f)$ for the cases where $(n, f) = (6, 2)$ and $(3, 1)$.

For more discussion, We conjecture that the upper bounds of asymptotic competitive ratio achieved in this paper would be optimal. Another conjecture is that $\lim_{\beta\to\frac{1}{2}} T(\beta) = 9$. In other words, when the number of byzantine robots is almost half of the total number, we can not do much better than the algorithm in [2].

Another interesting direction of this problem is how to improve the lower bounds. The hardness comes from the lack of methods to analysis the complicate behaviors of all robots at all time. For example, we even do not know whether all the robots should not stop at all time in every optimal algorithm to the searching problem with $(n, f) = (3, 1)$. However, a feasible way to the cases $(n, f) = (2f+1, f)$ is to consider a weaker version where all the faulty robots are crash fault. Some researchers have improve the lower bounds by this idea [19], and their results match the upper bound for crash robots. But for Byzantine version, the lower bounds are still not tight.

References

1. Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, 2002. 6

2. Anatole Beck. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964. 6, 19
3. Anatole Beck. More on the linear search problem. *Israel Journal of Mathematics*, 3(2):61–70, 1965. 6
4. Anatole Beck and DJ Newman. Yet more on the linear search problem. *Israel journal of mathematics*, 8(4):419–429, 1970. 6
5. Anatole Beck and Peter Warren. The return of the linear search problem. *Israel Journal of Mathematics*, 14(2):169–183, 1973. 6
6. Richard Bellman. An optimal search. *Siam Review*, 5(3):274, 1963. 6
7. Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012. 6
8. Marek Chrobak, Leszek Gąsieniec, Thomas Gorry, and Russell Martin. Group search on the line. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 164–176. Springer, 2015. 6
9. Huda Chuangpishit, Jurek Czyzowicz, Evangelos Kranakis, and Danny Krizanc. Rendezvous on a line by location-aware robots despite the presence of byzantine faults. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 70–83. Springer, 2017. 6
10. Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, Evangelos Kranakis, Danny Krizanc, and Najmeh Taleb. When patrolmen become corrupted: monitoring a graph using faulty mobile robots. *Algorithmica*, 79(3):925–940, 2017. 6
11. Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, and Sunil Shende. Search on a line by byzantine robots. In *27th International Symposium on Algorithms and Computation*, volume 27, 2016. 5, 6, 11
12. Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. Group search and evacuation. In *Distributed Computing by Mobile Entities*, pages 335–370. Springer, 2019. 6
13. Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, and Jaroslav Opatrny. Search on a line with faulty robots. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 405–414. ACM, 2016. 1, 2, 3, 4, 5, 6, 11, 17, 19
14. Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms (TALG)*, 11(1):1, 2014. 6
15. Shmuel Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27(1):17–30, 1974. 6
16. Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001. 6
17. Ming-Yang Kao, John H Reif, and Stephen R Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996. 6
18. Jon M Kleinberg. On-line search in a simple polygon. In *SODA*, volume 94, pages 8–15, 1994. 6
19. Andrey Kupavskii and Emo Welzl. Lower bounds for searching robots, some faulty. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 447–453. ACM, 2018. 5, 6, 19
20. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. 6

21. Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996. 6
22. Sven Schuierer. Lower bounds in on-line geometric searching. *Computational Geometry*, 18(1):37–53, 2001. 6