Operations Research Forum (2022) 3:9
https://doi.org/10.1007/s43069-022-00122-2

ORIGINAL RESEARCH

A Symbolic Programming Approach to the Rendezvous Search Problem

Pierre Leone[1] · **Steve Alpern**[2]

Received: 17 March 2021 / Accepted: 5 January 2022

Abstract

In this paper we solve the rendezvous problem on the line with markers that can be dropped at chosen times when the initial distance D between the players is known. In the case of one marker, the M_1 game, the marker is held by player II at the start of the game and, once dropped and found by player I, indicates in which direction player I must move. In the case of two markers, the M_2 game, each player holds one and the dropping times may differ. There is uncertainty regarding the problem initial configuration, and the goal is to minimize the expected rendezvous time that we call the rendezvous value (of the game) denoted R_1 and R_2 for the M_1 and M_2 games respectively. We present an algorithm that computes exactly the rendezvous value of the M_1 game as a function of the dropping time z, i.e. $z \mapsto R_1(z)$. Then we show that the function $R_1(z)$ is locally an affine function and we compute the parameters of the local representations of $R_1(z)$. Finally, the rendezvous value of the game $R_1 = min_z R_1(z)$ and the optimal dropping times can be determined with the expression of $R_1(z)$. The same proceeding can be extended to apply to the problem M_2. Symbolic execution of programs is a classical technique of program testing in computer science, see King [1] for the pioneering work. In this work we adapt the symbolic execution technique to solve an optimization problem. To our knowledge this is the first time that this is attempted, in particular to deal with rendezvous problems.

Keywords Rendezvous search game on the line · Symbolic programming · Rendezvous with markers · Enumeration of optimal strategies

✉ Pierre Leone
pierre.leone@unige.ch

[1] Department of Computer Science, University of Geneva, Geneva, Switzerland

[2] Warwick Business School, University of Warwick, Coventry CV4 7AL, UK

Published online: 31 January 2022

1 Introduction

In the classical formulation of the rendezvous problem that we consider in this article, two players are moving along the infinite line and wish to meet as soon as possible. At the beginning of the game, the players know they are placed at a distance D apart. Players move along the line at speed 1 and rendezvous occurs at the time both players occupy the same position.

Players have no orientation reference. Initially, each player faces one side of the infinite line and can move Forward or Backward accordingly to the side faced. The resulting motion of a Forward move on the line may be to the left or to the right side with equal probability and the Backward move is to the opposite side. The left/right sides are fixed and correspond to the left/right sides as seen by an observer. The Forward/Backward directions are seen by the players who do not know whether it is left/right, see Fig. 1.

There are then 8 initial conditions obtained by compounding the Forward directions of both players and the two possibilities where player I starts on the left or on the right of player II. All starting positions are equiprobable, see Fig. 1. The goal of the players is to minimize the expected meeting time that is called the rendezvous value and denoted by R. This game is defined and solved in [2], we call it M_0.

In this paper, we assume that player II has a marker that can be dropped off on the line at a chosen time at the position where he is located at that time. Once the other player, player I, is at the dropping position we say that the marker is found and this indicates to player I the direction on which player II is. As in M_0 the game stops when the two players rendezvous, i.e. occupy the same position on the line. The open question is to determine at which time it is optimal for a player to drop off the marker and when to change the direction (Forward/Backward) of motion? We call this game M_1.

A strategy for a player is given by the direction of subsequent moves (Forward or Backward), the times at which the player changes the direction of the motion and, in addition for player II, the time at which the marker is dropped off. An optimal solution of the game is a strategy pair, one strategy for each player, such that the expected rendezvous time is minimal. We denote by $R_1(z)$ the minimal expected rendezvous time when player II drops off the marker at time z, and $R_1 = \min_z R_1(z)$ the rendezvous value of the game.

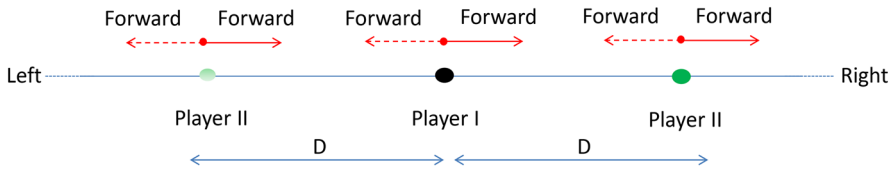

Fig. 1 The 8 possible initial conditions. Distance D between the players is known by both players. Forward direction may point in any direction and Backward points the opposite direction. By symmetry we can assume that player I Forward direction is always to the right side

The contributions of this paper are to present optimal strategy pairs to the M_1 game. Our approach extends our work in [3] where it is shown how to write a program to compute the function $z \mapsto R_1(z)$. Here, we show how the program can be modified to compute $0 = a_0 < a_1 < \ldots < a_n$ and affine functions $R_1^i(z) = b_i + c_i z$, $i = 1, \ldots, n$ such that $R_1^i(z) = R_1(z)$ for $z \in [a_{i-1}, a_i]$. The value of a_n is large enough to ensure that dropping off the marker at a later time is inefficient.

We also consider the game where player I and II have each one a marker and call this game M_2 and extend all the results to this setting.

Our approach is based on the symbolic execution of a program to compute these optimal strategy pairs. This work has been presented in a seminar[1] but has never been published yet. Up to our knowledge this approach is original.

2 Literature Review

The rendezvous search problem was proposed by Alpern in [4].This paradigm is prone to many variations. The references [5–10] provide a panorama of different settings and applications of the rendezvous problems. A first classification consider whether players move on continuous or discrete space. Time accommodates the same dichotomy. The first proper model of rendezvous search where the players can only meet at a discrete set of locations and time is discrete, is analysed in [11] and solved later for three locations in [12]. In this work we focus on models with continuous time and space and particularly on models where the motion happens along a one dimensional space (a line, a ring).

The rendezvous search problem for continuous space and time, including the infinite line, was first introduced in [13]. The player-asymmetric form of the problem (used in this paper), where players can adopt distinct strategies, was introduced in [2]. The corresponding player-symmetric problem on the line was developed in [14] and successively in [15, 16], and [17]. These papers assumed that the initial distance between the players on the line was known. The problem version where the initial distance between the players is unknown was studied in [18–20] and [21]. It is relevant that even in its simplest form the player-symmetric problem is still unsolved.

In [22], the authors allowed the players to leave markers at their starting points (at time 0). Our work here extends their results.

The continuous rendezvous problem has also been extensively studied on finite networks, the unit interval and circle [10, 23, 24]; arbitrary networks [25]; planar grids [26, 27]; and the star graph [28]. We have no knowledge of such works including the use of markers.

The rendezvous problem is an abstract problem that has some applications. In communication systems [29], robotic applications with different models: communication with faults [30], with asymmetric clocks [31], with asymmetric speeds [32].

[1] Search-and-Rescue rendezvous, The 18th International Symposium on Dynamic Games and Applications, Grenoble, France, July 9-12, (2018).

The rendezvous problem on the line may be motivated by the problem of two players lost on an island. A strategy to rendezvous for the players is first to reach the coastline and then apply a strategy for solving the rendezvous problem on the line, i.e. the players move in following the coastline. The line topology occurs naturally when the problem for the player is to escape a building by following a line on the floor [33, 34].

When the players are distinguishable, i.e. the player-asymmetric version of the game [14], the problem is said to have a universal strategy: One of the player is still while the other player searches for him. This strategy reduces rendezvous problems to search problems and is usually called *wait-for-mummy* strategy [6]. Hence, the optimal search strategy provide upper bounds on the asymmetric rendezvous value. The settings of search problems admit many variations as well [6, 7, 35, 36].

This paper is built upon the work done in [3] where it is shown how to solve the M_1 and M_2 problems by simulating the strategy pairs when the marker dropping time is known. The contribution of [3] consists in reducing the space of possibly optimal strategy pairs to a finite space (given that the dropping times are known). It is then possible to find the optimal strategy pairs by enumerating all of them and use a computer simulation to compare their performance. The algorithms in [3] compute the functions $z \mapsto R_1(z)$ and $(z_1, z_2) \mapsto R_2(z_1, z_2)$ that compute the minimal expected rendezvous time given the dropping time(s). Here, we extend this work to compute the minimal expected rendezvous time and compute the optimal dropping time(s).

3 Structure of the Paper

We describe the contribution of this paper in finding the optimal strategy pairs the M_1 and M_2 problems. We focus on the M_1 problem. The M_2 problem is handled similarly but at the price of more work. We distinguish a technical contribution (symbolic programming approach) and the contribution of solving the M_1 and M_2 problems (the optimal strategy pairs).

In Section 4 we present the rendezvous problem on the line in its classical form (without markers) and call it M_0. We show that finding the optimal strategy pairs can be done simply by enumerating all strategy pairs. This leads to a simple recursive program shown in Algorithm 1.

In Section 5 we introduce the problem M_1 where player II has a marker. It is shown how the previous algorithm of Section 4 can be extended to handle the marker of player II. This leads to an algorithm to compute the function $z \mapsto R_1(z)$. This algorithm takes as input the dropping time and enumerates all strategy pairs to output the ones minimizing the rendezvous time.

In Section 6 we extend the algorithm of Section 5 to compute $0 = a_0 < a_1 < \ldots < a_n$ and affine functions $R_1^i(z) = b_i + c_i z$, $i = 1, \ldots, n$ such that $R_1^i(z) = R_1(z)$ for $z \in [a_{i-1}, a_i]$. Because it happens that there are a finite number of such affine functions this program leads to the computation of the optimal dropping times and the optimal strategy pairs.

In the Apprendix Section 1 we describe how the formal computations can be extended to the problem M_2.

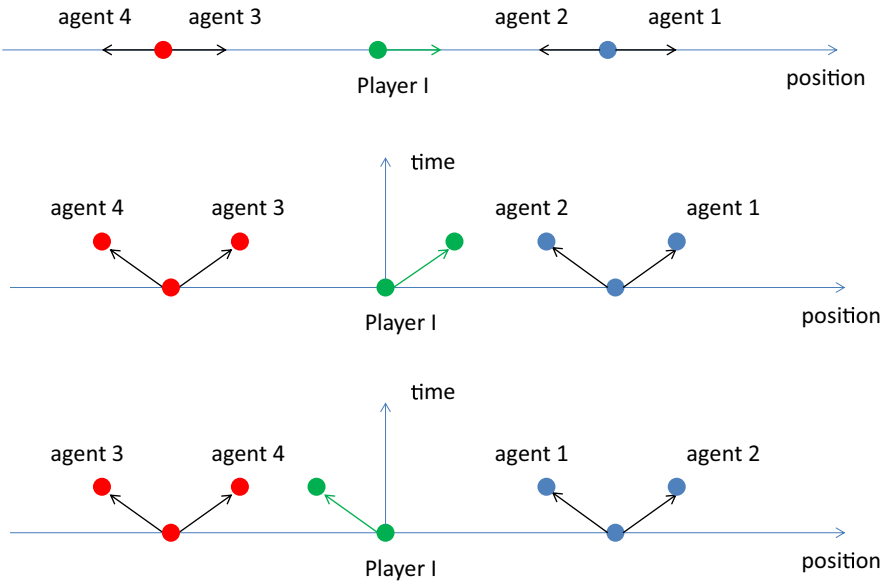

Fig. 2 Top: The derandomized initial condition with the 4 agents and player I Forward direction being unique. Middle: The motion resulting from a Forward move by player I and player II, i.e. the Forward move of player II results in the 4 moves of the agents. The motions are displayed in a time-position coordinate system (the speed of motion is 1). Bottom: The partial motion resulting from a Backward move by player I and player II. The motions are displayed in a time-position coordinate system (the speed of motion is 1)

4 Presentation and Solution of the M_0 Problem

The two players, player I and II, are placed at a known distance D apart and move on the infinite line. At the time when the two players occupy the same position rendezvous occurs and the game stops. The speed of motion is bounded, say by 1, to ensure a non zero rendezvous time. Proposition 3 of [3] shows that it does not restrict the generality to assume the speed of motion (maximal) equal to 1, i.e. moving at a lower speed is never optimal. There are two kinds of uncertainty:

- The players do not know whether the other player is on the left or on the right.
- Players move Forward or Backward but do not know whether these moves are to the left or to the right (only that a Backward move is opposite to a Forward one).

The 8 possible initial configurations are equiprobable and illustrated in Fig. 1. In fact, because of the symmetry of the problem we can assume that player I Forward direction always points in the same direction (the right) reducing to 4 the number of random initial configurations, see the top of Fig. 2. To illustrate, in case the two players move in their Forward direction, with probability 1/2 they are both moving to the right, hence the inter-distance is unchanged by the motion. With probability 1/4 they are moving towards each other, the inter-distance decreases by 2δ after a motion of duration δ. Finally, with probability 1/4 they are moving apart from each other, the inter-distance increases by 2δ after a motion of duration δ.

It is usual to derandomize the problem and make it deterministic. To play the role of player II we introduce 4 agents, two of them initially placed on the left of player I and two on the right. The Forward directions of two agents at the same initial location are opposite, see the top of Fig. 2. With this representation, when the strategy of player II specifies a move, say Forward, it results in 4 deterministic moves by each agent in their respective Forward direction. The game now is that player I must rendezvous with the 4 agents of player II in minimum average time, the sum of the rendezvous time of player I with the agents being divided by 4. Figure 2 illustrates the derandomized initial condition with the 4 agents as well as the motions resulting from a Forward and a Backward move. The derandomization of the game is used as well in search game [6]. Here we introduce a name for the derandomized version of M_0.

Notation In the text we refer to the derandomized version of M_0 as the MD_0-game, where D stands for deterministic or derandomized.

To recapitulate, the original game is M_0 and in this game player I plays with player II. The game stops when player I rendezvous with player II. In MD_0 player I plays with the four agents of player II and the goal is to rendezvous with all of them. In M_0 player I can be seen as to play with only one of the agents of player II chosen randomly with uniform probability (1/4). The goal is to minimize the expected or average rendezvous time in M_0, MD_0 respectively.

In MD_0 player strategies are successions of Backward (B) or Forward (F) moves and are displayed here as FBFB, for instance, to mean that the player is moving Forward, then Backward, then Forward, then Backward. Strategies of players I and II can be different and we write a strategy pair as (FBFB - FFBB), for instance, to display the strategy of player I (first) and of player II (second). Actually, this particular strategy is optimal [2] and shown in Fig. 3. A move is executed by the players at speed 1 and until player I rendezvous with one of the agents. Rendezvous occurs when both player I and the agent are located at the same position.

There are a finite number of strategy pairs. Indeed, in MD_0 player I must rendezvous with the 4 agents. A given strategy pair is equivalent to fix the order of player I rendezvous with the agents. For instance, this order may be $\{2, 3, 1, 4\}$, leading player I to move first F in the direction of agent 2, then B in the direction of agent 3, then F in the direction of agent 1 and finally B in the direction of agent 4. Player II corresponding strategy is F making the agent 2 moving towards player I, then F making agent 3 moving towards player I, then B making agent 1 moving towards player I, then B making agent 4 moving towards player I, see Fig. 3.

For the rendezvous times we use the notation t^i for[2] the rendezvous time of player I with agent i. If at time t players I and II set their strategies such that player I rendezvous with agent i the time needed to rendezvous is $\frac{d_i}{2}$ where d_i is the distance between player I and agent i at time t. The rendezvous then occurs at time $t^i = t + \frac{d_i}{2}$. The times $t_1 \le t_2 \le t_3 \le t_4$ denote the rendezvous times t^i by increasing order, see the illustration in Fig. 3.

[2] We apologize if the notation may be confused with t into the power of i.

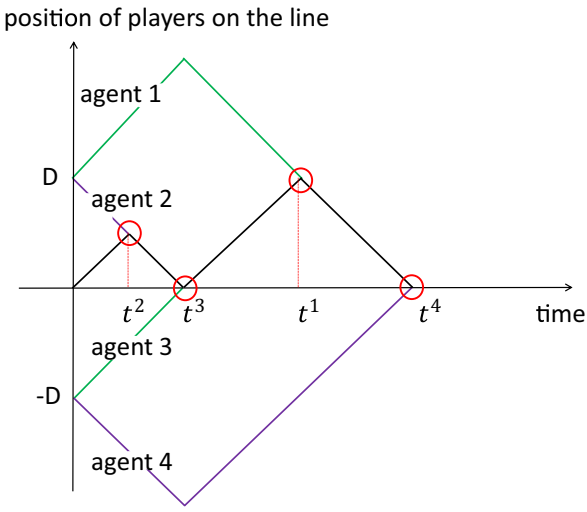

Fig. 3 The trajectories resulting from the strategy pair (FBFB - FFBB). Times t^i denotes the rendezvous of player I with agent i. Sorted in increasing order the rendezvous times are $t_1 = t^2 = \frac{D}{2}$, $t_2 = t^3 = D$, $t_3 = t^1 = 2D$, $t_4 = t^4 = 4D$. The circles highlight the rendezvous between player I and the agents. Notice the symmetrical strategy pairs: Player II can apply BBFF (instead of FFBB) where agent 1 swaps his path with agent 2 and agent 3 swaps his path with agent 4. Player I can follow BFBF (instead of FBFB)

Hence, a strategy pair is given by a permutation σ of the set $\{1, 2, 3, 4\}$ that specifies that player I rendezvous with agents $\sigma(1), \sigma(2), \sigma(3), \sigma(4)$ one after the other. With the notation introduced before we get $t_1 = t^{\sigma(1)}$, $t_2 = t^{\sigma(2)}$, $t_3 = t^{\sigma(3)}$, $t_4 = t^{\sigma(4)}$.

The goal of the players is to minimize the average rendezvous time given by

$$R = \frac{t_1 + t_2 + t_3 + t_4}{4} = \frac{t^1 + t^2 + t^3 + t^4}{4}.$$

Because there are 24 possible permutations[3] σ, a computer program can execute all strategies and output the strategies with minimal average rendezvous time. A procedure that enumerates all strategy pairs is shown in Algorithm 1. This procedure is already discussed but not given explicitly in [3]. Running this program is an alternative proof to the one in [2] and the strategy pair (FBFB - FFBB) appears to be optimal, see Fig. 3.

The initial distance D is set to 1 in Algorithm 1 for computation. To obtain the results in full generality the rendezvous times have to be multiplied by D (indeterminate), the rendezvous times being linear functions of the intial distance. Moreover, notice that Algorithm 1 makes only use of addition or division by 2 on variables. Hence, if the initial distance D is chosen of the form $D = 2^N$, and N is large enough, then all the values computed are integers. The algorithm can then be executed without round-off errors. This remark is also valid for the M_1 and M_2 games.

[3] By symmetry we can assume that the Forward direction of player I is to the right side and that player I and II start with a Forward move. Player I always rendezvous with agent 2 first and this reduces the number of permutations to 6.

procedure CheckAllStrategy($d_1, d_2, d_3, d_4, t, p_1, p_2, total$)
if $(d_1 > 0)$ **then**
  CheckAllStrategy($0, d_2, d_3 \tilde{+} d_1, d_4, t + \frac{d_1}{2}, p_1 + 'F', p_2 + 'B', total + t + \frac{d_1}{2}$)
end
if $(d_2 > 0)$ **then**
  CheckAllStrategy($d_1, 0, d_3, d_4 \tilde{+} d_2, t + \frac{d_2}{2}, p_1 + 'F', p_2 + 'F', total + t + \frac{d_2}{2}$)
end
if $(d_3 > 0)$ **then**
  CheckAllStrategy($d_1 \tilde{+} d_3, d_2, 0, d_4, t + \frac{d_3}{2}, p_1 + 'B', p_2 + 'F', total + t + \frac{d_3}{2}$)
end
if $(d_4 > 0)$ **then**
  CheckAllStrategy($d_1, d_2 \tilde{+} d_4, d_3, 0, t + \frac{d_4}{2}, p_1 + 'B', p_2 + 'B', total + t + \frac{d_4}{2}$)
end
if $(d_1 + d_2 + d_3 + d_4 == 0)$ **then**
  print("final time t; player I strategy p_1; player II strategy p_2")
  print("average rendezvous time total/4")
end

Algorithm 1 Pseudo-code of the program that enumerates the 24 strategy pairs of the rendezvous problem on the line. The recursive procedure prints out all the checked strategy pairs and the total time. The parameters di are the distances from player I to agent *i* of player II, *t* is the current time, p_1, p_2 are strings initially empty that contain the players' strategies, i.e. sequence of Forward and Backward moves and total is the sum of the rendezvous times. The agent numbering is according to Fig. 2. We use an operator denoted $\tilde{+}$, i.e. $x \tilde{+} y$ whose semantic is: **if** $x > 0$ **then** $x + y$ **else** 0. The operator $\tilde{+}$ is used instead of + because a null distance d_i indicates that player I and agent *i* have met and, if null, di must not be updated. To solve the problem call CheckAllStrategy(1,1,1,1,0,"","",0).

5 Presentation of the M_1 Game

In this section we introduce the new game M_1 which is a generalization of the game introduced in [22]. The setting is the same as for the M_0 game except that player II has a marker that can be dropped at any chosen time. The derandomization tool that we discuss for M_0 in Section 4 is still useful. Player II random initial configurations are replaced by 4 agents, each one having a marker at its disposal (the agents drop off the marker at the same time). Again we give a name to the derandomized version of M_1.

Notation In the text we refer to the derandomized version of M_1 as the MD_1-game, where D stands for deterministic or derandomized.

In MD_1 strategies for players are still given by a succession of Forward and Backward moves and the strategies for player II are extended with the time z when the marker has to be dropped off. For instance a strategy for player I is FFFBB while a strategy for player II is z;FBFFB, the z denoting the dropping time.

The execution of a strategy pair goes in the following way. The players start to move at speed 1 in the direction given by the first letter of their strategy (F or B). The direction of the player's motion switches according to the next letter in the strategy each time one of the following events occurs

1. Player I rendezvous with agent i, this defines time t^i,
2. the current time t equals the dropping time z of player II, i.e. $t = z$,
3. player I finds the marker of an agent, this can occur only if the current time t satisfies $t \geq z$.

This shows that if the dropping time z is known a strategy for a player consists of at most 5 moves (F or B, 4 rendezvous or markers found (events 1. or 3.) plus 1 for dropping the marker. Hence, there is at most $32^2 = 1024$ strategy pairs that can be all simulated to find the optimal ones. The strategies can be enumerated and simulated to find out the optimal ones. We emphasize that this does not completely solve the problem since the dropping time z is not known, as assumed here, and must be chosen optimally. This will be discussed in the next Section 6.

It is important to describe what happens after player I finds the marker of agent i. The first remark is that the direction that player I follows when he arrives to the marker location (F or B) is the direction that player I has to follow to rendezvous with agent i. This is because agent i cannot be on the other side of the line that has been crossed by player I (rendezvous would have occurred). Hence, after finding the marker the strategy becomes immaterial for player I to rendezvous with agent i. However, player I has to rendezvous with the other agents and for this he needs to follow the strategy. This means that after finding a marker, player I has to follow two paths simultaneously, player I splits into two subplayers. One goes on to rendezvous with agent i and one continues according to the strategy. This appears in Fig. 4 at time $\frac{2}{3}$ when player I finds the marker of agent 2. At that time, one subplayer of player I continues in the Forward direction to rendezvous with agent 2 (discarding the strategy) and the other subplayer follows the strategy and goes Backward. The fact that player I splits after finding a marker is an artefact of the derandomization tool.

Actually, in the M_1 game, player I plays with player II (with one of the agents at a time). Hence, if player I finds the marker he continues in the same direction ignoring the strategy until meeting player II. This happens at time $\frac{2}{3}$ in Fig. 4 if player II initial configuration is the one of agent 2. If the marker is not found at time $\frac{2}{3}$, player I follows the strategy and switch to the next direction. This happens in Fig. 4 at time $\frac{2}{3}$ if player II initial configuration is the one of agent 1, 3 or 4. In MD_1 player I has to split because he is playing with all 4 agents simultaneously, hence he behaves as if the marker is found and as if the marker is not found (see Fig. 4).

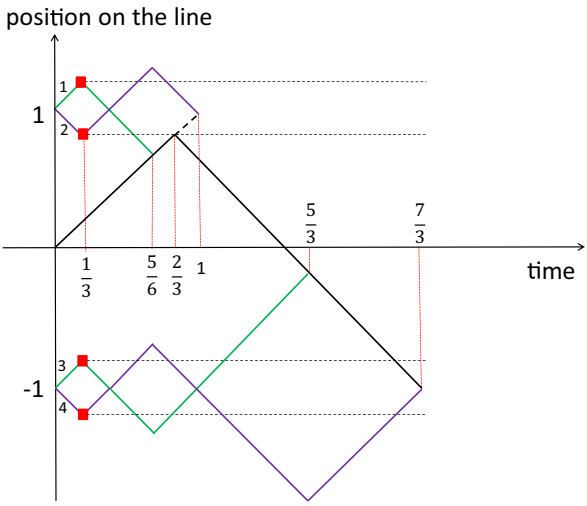

Fig. 4 Graphical illustration of the strategy $(FFFBB-\frac{1}{3};FBFFB)$ with $D=1$. The times are $t_1=t^1=\frac{5}{6}$,$t_2=t^2=1, t_3=t^3=\frac{5}{3}$, $t_4=t^4=\frac{7}{3}$. The paths are labeled with the number of the corresponding agent. The horizontal dotted lines indicate the existence of the marker at the position. Players switch direction according to the next letter in their strategy at times $\frac{1}{3}$, $\frac{5}{6}$, $\frac{2}{3}$, 1, $\frac{5}{3}$

To convince the reader that a program that simulates the execution of a strategy pair can be written we describe the execution of the strategy pair $(FFFBB-\frac{1}{3};FBFFB)$[4] with $z=\frac{1}{3}$ and $D=1$, which is plotted on Fig. 4. Both players start with a forward move (F) which means that player I and agent 2 rendezvous if the motion lasts long enough, i.e. the rendezvous time occurs before the dropping time z. The time of motion ensuring rendezvous is $\frac{1}{2}$ because the distance separating player I and agent 2 is 1 and both move at speed 1. The dropping time $z=\frac{1}{3}<\frac{1}{2}$ occurs before the rendezvous. At this time, Player I and II switch to the next direction in their strategy, F and B respectively. At time $\frac{5}{6}$, player I rendezvous with agent 1 and players I and II switch to the 3rd direction in their strategy F and F respectively. At time $\frac{2}{3}$ player I finds the marker of agent 2. At this point, player I splits in two subplayers, one subplayer keeps going Forward until rendezvous with agent 2 occurs (at time 1) and the other subplayer switches to the next direction in his strategy (follows the strategy) with a B move. Player II follows the strategy with the 4th direction which is a F move. At time 1, one subplayer of player I rendezvous with agent 2. This event stops the subplayer of player I and agent 2.

The next event occurs at time $\frac{5}{3}$ when player I rendezvous with agent 3. After the rendezvous player I continues with a Backward move (B) and player II (agent 4) with a Backward move (B).

To summarize, rendezvous occur at times $\frac{5}{6}$, 1, $\frac{5}{3}$ and $\frac{7}{3}$. The average rendezvous time R is

$$R_1(FFFBB-\frac{1}{3};FBFFB)=\frac{35}{24}. \tag{1}$$

If the dropping time is known, the execution (simulation) of a strategy pair is illustrated by Algorithm 2. The algorithm takes as input a strategy pair and a fixed concrete value for the dropping time z. Calling Algorithm 2 for all 1024 strategy

[4] Player I follows strategy FFFBB and player II FBFFB with dropping time $z=\frac{1}{3}$.

pairs and comparing the results leads to the identification of optimal strategy pairs for a given (fixed) dropping time z, i.e. the computation of the function $z \mapsto R_1(z)$.

Input: A strategy for player I, e.g. $(FFFBB)$; a strategy for player II with dropping time, e.g. $(z; FBFFB)$.
Output: The average of the four rendezvous times $total/4$.
set current time $t = 0$.
set total rendezvous time $total = 0$
set MovePlayerI and MovePlayerII to the first letter of their strategy (F or B)
while *current time* $t \leq z$ **do**
    execute the next move accordingly to MovePlayerI and MovePlayerII and stop when rendezvous occurs or the current time t equals the dropping time z
    if t==z **then**
        drop the marker of player II - one marker for each agent dropped at the current agent's locations
        set MovePlayerI and MovePlayerII to the next letter in their strategy
    end
    if *rendezvous occurs* **then**
        set MovePlayerI and MovePlayerII to the next letter in their strategy
        $total = total + t$
    end
end
while *There is one agent to rendezvous with* **do**
    execute the next move: stop if rendezvous occurs or player I finds a marker
    update current time t
    if *rendezvous occurs* **then**
        set MovePlayerI and MovePlayerII to the next letter in their strategy
        $total = total + t$
    end
    if *player I finds a marker* **then**
        split in two player I
        one double keeps the same direction until rendezvous occurs with the agent that dropped the marker and after rendezvous update the sum of rendezvous times $total = total + t$
        one double continues to apply the strategy: set MovePlayerI to the next letter in the strategy of player I
        set MovePlayerII to the next letter in the strategy of player II
    end
end
Output the average rendezvous time: $total/4$

Algorithm 2 Pseudo code of the simulation of a strategy pair that leads to the computation of the average rendezvous time for this strategy pair. Notice, the dropping time z is a concrete value. example of a program is $P : z \rightarrow < v = z/2, w = v + z, x = 2 + w, return\ x >$ ($v, w, x\ \epsilon\ \mathbb{R}$ are local variables). With P(2) denoting the value returned by the program P if the formal argument z is given the value 2 we obtain in our example *P(2) = 5*. Notice that here we assume that the program is computing with values in $\mathbb{R}$. In practice it computes with types like *integer, double, etc*. but the reasoning below remains valid.

6 Computing the Exact Solution of the MD_1 Game

The difficulty to solve the MD_1 game is that the dropping time z is a continuous variable. To resolve this issue, we modify the Algorithm 2 such that the dropping z is no longer a fixed (concrete) value but a symbolic variable (an unknown).

The important remark is that in Algorithm 2, the numerical operations applied to times and distances are:

- Division by 2, to compute the time it takes for player I and an agent to rendezvous when both are moving towards each other (each one is moving at speed 1),
- Addition, to compute the current or total times or the distance from player I to a marker or an agent,
- Subtraction, to compute the distance from player I to a marker or an agent.

Lemma 1 *The operations of division by* 2 ($/2$), *addition* ($+$) *and subtraction* ($-$) *preserve the set of affine functions* $\mathbb{S}$ *defined by*

$$\mathbb{S} = \left\{ a + bz \,\middle|\, a, b \in \mathbb{R} \text{ and } z \text{ is indeterminate} \right\}. \tag{2}$$

This means that these operations can be seen as operations on $\mathbb{S}$, *i.e.* $/2 : \mathbb{S} \rightarrow \mathbb{S}$, $+ : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{S}, - : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{S}$.

Affine functions are defined by their coefficients, to simplify the notation we sometimes denote by (a, b) *the affine function* $a + bz$.

Proof These operations are defined on $\mathbb{S}$ as

- $(a, b)/2 = (a/2, b/2)$ or equivalently $(a + bz)/2 = a/2 + b/2z$,
- $(a, b) + (a', b') = (a + a', b + b')$ or equivalently $a + bz + a' + b'z = a + a' + (b + b')z$,
- $(a, b) - (a', b') = (a - a', b - b')$ or equivalently $(a + bz) - (a' + b'z) = a - a' + (b - b')z$.

We see from the definition that these operations preserve $\mathbb{S}$, i.e are functions $\mathbb{S} \rightarrow \mathbb{S}$ or $\mathbb{S} \times \mathbb{S} \rightarrow \mathbb{S}$.

An operation that does not preserve $\mathbb{S}$ is the multiplication since by multiplying to affine functions we get a polynomial of order 2.

A program is composed of a sequence of operations. We denote a program P composed of n instructions and accepting a formal argument z by $P : z \rightarrow < a_1, a_2, \ldots, a_n >$. The program returns a value using the operation *return*. An example of a program is $P : z \rightarrow < v = z/2, w = v + z, x = 2 + w, return\ x >$ ($v, w, x \in \mathbb{R}$ are local variables). With $P(2)$ denoting the value returned by the program P if the formal argument z is given the value 2 we obtain in our example $P(2) = 5$. Notice that here we assume that the program is computing with values in $\mathbb{R}$. In practice it computes with types like *integer, double, etc.* but the reasoning below remains valid.

Lemma 2 *We consider a program composed of a sequence of operations restricted to be*: *division by* 2 (/2), *addition* (+) *and subtraction* (−). *We assume that this program takes a real input and we denote z the parameter or formal argument. We denote* $z \mapsto f(z)$ *the function computed by the program. The function f*(*z*) *is affine*.

Proof To prove the Lemma we change our point of view on the program execution. We take the program P of our example as support of the proof. We consider that the formal argument is not given a value but compute the result of the program algebraically, using z as an indeterminate. Because in that case the input of the program is the affine function $z \rightarrow$ z and each operation applied by the program preserves the affine functions by Lemma 1, the result output by P is an affine function. For the program P in our example we obtain $v = z/2, w = \frac{3}{2}z, x = 2 + \frac{3}{2}z$, showing that the value returned by the program can well be expressed as an affine function. It can be checked that if we assign the value 2 to z in the expression $2 + \frac{3}{2}z$ the value returned is 5 again.

In order to turn a program like P to compute with variables and constants in the set of affine functions $\mathbb{S}$ instead of real values $\mathbb{R}$ the variables and constants have to be implemented as pairs of real values, (a, b) denoting the affine function $a + bz$. The operations of division by 2 (/2), addition (+) and subtraction (−) are then implemented as shown in the proof of Lemma 1. With this formalism, our example program P becomes $P' : (0, 1) \rightarrow < v = (0, 1)/2, w = v + (0, 1), x = (2, 0) + w, return\ x >$ that returns $(2, \frac{3}{2})$ which is the expected affine function.

We are almost done. The last operation that Algorithm 2 makes use and that we have not defined on $\mathbb{S} \times \mathbb{S}$ is **the comparison**. Given $(a, b), (a', b') \in \mathbb{S}$ it is needed to decide which one is the smaller, for instance, to decide whether a rendezvous occurs before the dropping time.

The first idea is to say that an affine function (a, b) is smaller than (a', b') if the values computed are always smaller, i.e. $\forall z \in \mathbb{R}$, $a + bz < a' + b'z$, which is not a total order. To get a total order we need to constrain the domain of the affine function (the values that can be assigned to z). In our case, the values of z are dropping times and are assumed to be positive. So we come with the following definition for the comparison operator $<_{\mathbb{S}}$.

Definition 1

$$\begin{aligned} a + bz <_{\mathbb{S}} a' + b'z \Longleftrightarrow & \big((a < a') \text{ and } (b > b') \text{ and } z < \frac{a' - a}{b - b'}\big) \text{ or} \\ & \big((a < a') \text{ and } (b < b') \text{ and } z > \frac{a' - a}{b - b'}\big) \text{ or} \\ & \big((a < a') \text{ and } (b = b')\big) \text{ or} \\ & \big((a = a') \text{ and } (b < b')\big). \end{aligned}$$

Notice that in the case where $a < a'$ and $b < b'$ the condition on z, $z > \frac{a'-a}{b-b'}$ is always satisfied since z is assumed positive.

To be operational this definition requires that the program keeps track of the smallest bound computed along the computations. The affine function computed by the program will be the valid result only for values of z smaller than this bound. Let us describe this on the following example

$$\begin{aligned} P^F : (0,1) \rightarrow < & \textbf{ if } (0,1) < (1,-2) \textbf{ then } \{v = (0,1)/2\} \\ & \textbf{else } \{v = (0,0)\}, w = v + (2,1), \textit{return } w > \end{aligned}$$

At the start of the execution of P^F the value of the bound of z, denoted by z_{bound} is infinite. The test $(0,1) < (1,-2)$ happens to be true if z is bounded by $\frac{1}{3}$. The condition is then true and z_{bound} is set to $z_{bound} = \min(z_{bound}, \frac{1}{3}) = \frac{1}{3}$. Then the program computes $v = (0,1)/2$ and $w = (2,2)$. The results is $(2,2)$ **and** $z < \frac{1}{3}$.

The discussion in this section up to this point is summarized, for emphasis, in the following proposition.

Proposition 1 *Consider an algorithm that computes with data in* $\mathbb{S}$ *and whose operations are divisions by* 2 (*noted* /2), *additions*, *subtractions and tests using the order relation* $<_{\mathbb{S}}$ *of Definition* 1. *Then*, *the program result is in* $\mathbb{S}$ *and is correct for values of* z *in the interval* $[0, z_{bound}]$ *where* z_{bound} *is the smallest bound computed by the program when comparing members of* $\mathbb{S}$ *with Definition* 1. *Computing a real value by substituting a value for* z *in the result leads to the same numerical value as if the program was written to compute with real values*.

To compute the result of the program for values larger than z_{bound} we now consider values of z of the form $z = z_{bound} + z'$ with $z' \geq 0$. This leads to a modification of the order relation $<_{\mathbb{S}}$ which is denoted by $\widetilde{<}_{\mathbb{S}}$ and is the order relation used in the program. In the definition we use a general *offset* instead of the particular z_{bound}.

Definition 2

$$\begin{aligned} & a + bz \,\widetilde{<}_{\mathbb{S}}\, a' + b'z \Longleftrightarrow a + b(\mathit{offset} + z') <_{\mathbb{S}} a' + b'(\mathit{offset} + z') \\ & \Longleftrightarrow (a + b \cdot \mathit{offset}) + bz' <_{\mathbb{S}} (a' + b' \cdot \mathit{offset}) + b'z'. \end{aligned}$$

With this definition we continue the execution of the program P^F with *offset* $= \frac{1}{3}$. The test leads to check $(0,1)\widetilde{<}_{\mathbb{S}}(1,-2)$ which by Definition 2 leads to $(\frac{1}{3},1)<_{\mathbb{S}}(\frac{1}{3},-2)$, which by Definition 1 is false. Hence P^F computes $v=(0,0)$ and returns $w=(2,1)$. In computing the value of the test ($<_{\mathbb{S}}$), because the result is false, the bound on z' is not updated. Hence the results is valid for $z' \in [0,\infty[$, or $z \in [\frac{1}{3},\infty[$.

Executing P^F twice, we have obtained the formal expression of the function $z \mapsto P''(z)$ with $P'' : z \to <$ **if** $z < (1-2z)$ **then** $\{v = z/2\}$ **else** $\{v = 0\}, w = v + 2 + z$, *return* $w >$.

$$P^F(z) = \begin{cases} 2 + \frac{3}{2}z & \text{if } 0 \le z < \frac{1}{3} \\ 2 + z & \text{if } z \ge \frac{1}{3} \end{cases}$$

We have now collected all the material to prove one of the main result of this article.

Theorem 1 (Single Marker Game) *The optimal strategy pair for the rendezvous problem on the line with a single marker* (*possessed by II*) *is as follows*:

$$(FFFBB - D/4; FBFFB),$$

where the times at which direction may change are

$$D/4,\ 3D/4,\ 7D/4,\ 5D/2.$$

The rendezvous times are

$$t^1 = t_1 = 3D/4,\ t^2 = t_2 = D,\ t^3 = t_3 = 7D/4,\ t^4 = t_4 = 5D/2.$$

The rendezvous value R_1 (*the average of the rendezvous times*) *for this problem is given by*

$$R_1 = 3D/2.$$

Proof It is shown in Section 5 that if the dropping time z is known it is possible to write a program that computes the function $z \mapsto R_1(z)$. This program makes use of only the operations of Lemma 1, division by 2 (/2), addition (+) and subtraction (−). As we have seen in this Section 6, such a program can be turned to compute symbolically. The proof consists in the execution this program. At the start the value of the offset is *offset* $= 0$. The algorithm checks all strategy pairs and outputs that the rendezvous value of the game is $13/8 - 1/2z$ for $z \in [0, 1/6]$. The strategy pairs that are identified as optimal are $FFFBB - FBFFB$, $FFFBB - BFBBF$, $BBBFF - FBFFB$ and $BBBFF - BFBBF$. These 4 strategy pairs lead to the same figure as shown in Fig. 7. Indeed, if player II plays *BFBBF* instead of *FBFFB* this amounts to exchange agent 1 with 2 and agents 3 with 4. Similarly, if player I plays *BBBFF* instead of *FFFBB* this amounts to exchange agent 1 with 4 and agent 2 with 3. The strategy pairs that have in common such symmetry are considered as similar.

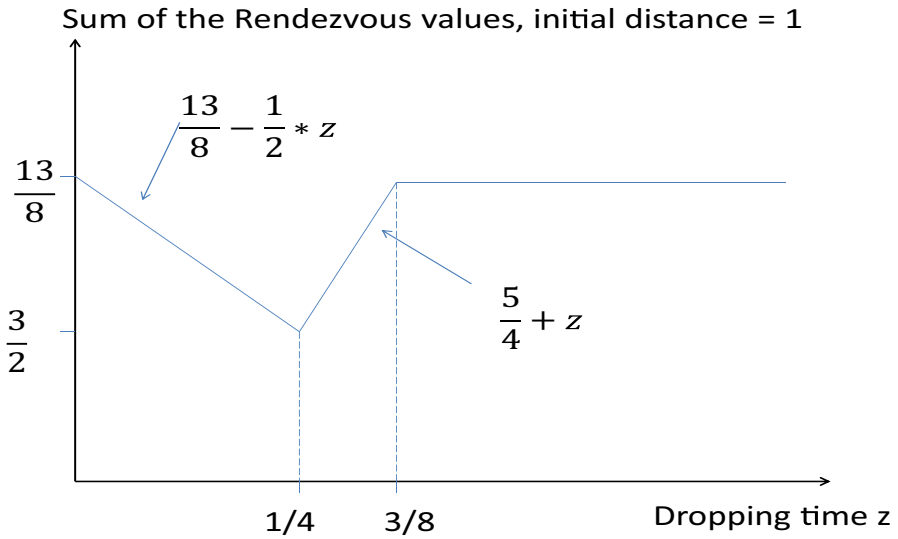

Fig. 5 Plot of the function $z \mapsto R_1(z)$ with $D = 1$. The strategy pairs leading to $R_1(z)$ are plotted in Fig. 7, on the left for $0 \le z < D/4$ and on the right for $D/4 \le z < 3/8$. For $z \ge 3/8$ the marker is useless and the optimal strategy pair is the one solving M_0 and plotted in Fig. 3

To continue the computations, we set the offset $offset = 1/6$ to cover larger values of the dropping time z, i.e. $z = offset + z'$, with $z' > 0$. For this second step we obtain that the rendezvous value of the game is $13/8 - 1/2z$ for $z \in [1/6, 1/4]$ and the optimal strategy pairs are the same as the ones identified for $z \in [0, 1/6]$ and plotted in Fig. 7. The value of the bound for z' is 1/12 ($z = \frac{1}{6} + z'$, hence we get the bound $1/6 + 1/12 = 1/4$ for z). We proceed similarly until the meaningful dropping

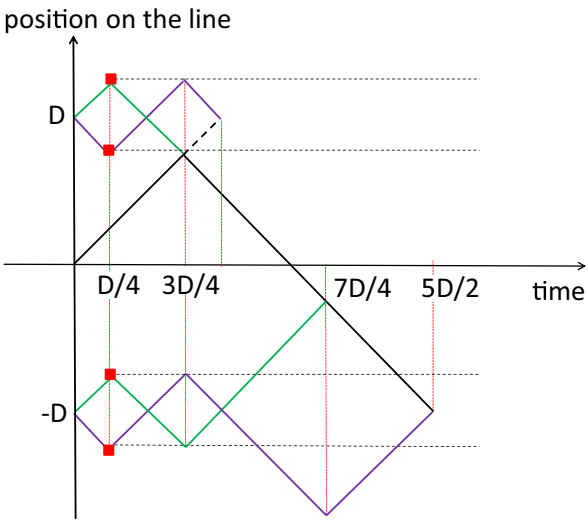

Fig. 6 Optimal strategy pair for the one marker game. Horizontal dotted lines indicate stationary location of dropped marker by various agents. Dashed line starting at time 3D/4 indicates I's motion upon finding marker

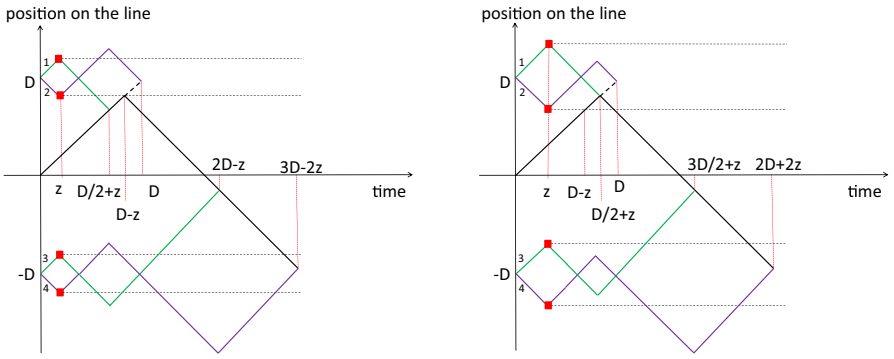

Fig. 7 Left:Graphical illustration of the strategy $(FFFBB-z;FBFFB)$, with $z < D/4$. The rendezvous times are $t_1 = t^1 = D/2 + z, t_2 = t^2 = D$, $t_3 = t^3 = 2D - z$, $t_4 = t^4 = 3D - 2z$. The paths are labelled with the number of the corresponding agent. The horizontal dotted lines indicate the existence of the marker at the position. Players switch direction according to the next letter in their strategy at times z, $D/2 + z$, $D - z$, $2D - z$. Right:Graphical illustration of the strategy $(FFFBB - z;FBBFB)$, with $D/4 < z < 3D/8$. The rendezvous times are $t_1 = t^1 = D/2 + z, t_2 = t^2 = D$, $t_3 = t^3 = 3D/2 + z$, $t_4 = t^4 = 2D + 2z$. The paths are labelled with the number of the corresponding agent. The horizontal dotted lines indicate the existence of the marker at the position. Players switch direction according to the next letter in their strategy at times z, $D - z$ $D/2 + z$, $3D/2 + z$

times are all covered. The function $z \mapsto R_1(z)$ is plotted in Fig. 5. We see that it is optimal for player II to drop off the marker at time $z = D/4$ and the optimal strategy pair is plotted in Fig. 6. In Fig. 7 we have plotted the optimal strategy pairs for $z < D/4$ and $1/4 \leq z < 3/8$. The optimal strategy pair for $z \geq 3/8$ is the one of M_0 plotted in Fig. 3.

7 Conclusion

The symbolic execution of program can be adapted to solve the M_2 problem as well. The program that computes the function $(z_1, z_2) \mapsto R_2(z_1, z_2)$ is described in [3]. This time, the symbolic program computes with affine functions in $\mathbb{S}_2$ of the form $a + bz_1 + cz_2$, see Section 1. Division by 2, addition and subtraction are defined in the natural way. To compare the elements of $\mathbb{S}_2$ requires more work than in $\mathbb{S}$. Moreover, to cover the 2-dimensional region where (z_1, z_2) take their values requires more *offset* than in the definition of $\widetilde{<}_{\mathbb{S}}$. This is presented in the Appendix 1. The execution of the symbolic program confirms that the strategy pairs stated in Theorem 9 of [3] are optimal. Notice that we observe that the rendezvous value of the M_2 game is the same as the one of the M_1 game. This shows that using a second marker provides more optimal strategy pairs but does not improve the expected rendezvous time.

In [37, 38], we have considered different rendezvous problems and we have shown how to reduce these rendezvous problems to families of linear programs. By

solving the LPs composing the family we compute the optimal dropping time z and identify the optimal strategy pairs of the associated rendezvous problem. However, with this approach the analysis of the function $z \mapsto R_1(z)$ cannot be done. With the symbolic approach suggested here, such an analysis is possible as we have shown in Section 6.

The main observation leading to the results is that the program that computes the function $z \mapsto R_1(z)$ makes only use of of division by 2 (/2), addition (+) and subtraction (−). This observation resonates with the fact some arithmetic theories that do not use multiplication are decidable. For instance, the Presburger arithmetic is a first-order theory on the integers that does not use the multiplication. It is then tempting as an open question to investigate how a rendezvous problem could be formulated as such or in more general arithmetic theory [39]. Such an approach would lead to a formulation of the rendezvous problems in formal logical terms and makes possible the use of efficient solvers [39, 40] to deal with more challenging rendezvous scenarios.

Computing the Exact Solution of the MD_2 Game

With two markers, the strategies are given by sequences of Forward or Backward moves and the dropping times of player I and player II. The game is derandomized similarly to M_0 and M_1 to define the MD_2 game where player I must rendezvous with the 4 agents of player II and the average rendezvous time is to be minimized. Players switch to the next direction in their strategy when one of the following events occurs:

1. The time player I rendezvous with agent i,
2. The time player I finds the marker of agent i,
3. The time agent i finds the marker of player I,
4. The dropping time of player I's marker, we denote z_1,
5. The dropping time of player II's marker, we denote z_2.

The strategy pairs are then given by $(z_1;XXXXXXXXXX - z_2;YYYYYYYYYY)$ where z_1 and z_2 are dropping times of players I and II respectively and the X's and Y's are sequences of Forward (F) or Backward (B) moves and the strategy of player I is given first. Each strategy has a maximal length of 10 symbols (F or B): 4 accounting to the events 1. or 2., 4 accounting to the events 1. or 3., 1 for event 4. and 1 for event 5. (after the last event the game stops and there is no need to switch to any direction). Hence, the number of strategy pairs is bounded by $1024 * 1024 = 1048576$ ($1024 = 2^{10}$) given that the dropping times z_1 and z_2 are known.

An algorithm similar to Algorithm 2 is then possible that given a strategy pair for players I and II (with z_1 and z_2 known) simulates the motions to find the average rendezvous time. By exhaustively testing at most 1048576 possible strategy pairs it is possible to compute $R_2(z_1, z_2)$ the average rendezvous time given the dropping times z_1, z_2. The next problem is again that z_1 and z_2 are continuous variables to be optimized.

Similarly to what we observed in Section 6 for the problem with one marker, the program that simulates all trajectories makes use of addition, subtraction and multiplication by constant (actually only division by 2). Hence, in the program these operators can be overloaded to compute with variables in $\mathbb{S}_2$, where

Definition 3

$$\mathbb{S}_2 = \left\{ a + bz_1 + cz_2 \;\middle|\; a, b, c \in \mathbb{R} \text{ and } z_1, z_2 \text{ are indeterminate} \right\}.$$

Addition, subtraction and multiplication by constant are defined similarly than for the M_1 problem, i.e.

- $(a + bz_1 + cz_2)k = (ak) + (bk)z_1 + (ck)z_2$, with k a constant in $\mathbb{R}$,
- $a + bz_1 + cz_2 + a' + b'z_1 + cz_2 = a + a' + (b + b')z_1 + (c + c')z_2$,
- $a + bz_1 + cz_2 - (a' + b'z_1 + c'z_2) = a - a' + (b - b')z_1 + (c - c')\, z_2$.

It is still needed to specify how distances (or times) are compared. This amounts to define a total order on $\mathbb{S}_2$ and we proceed with the definition of positive members of $\mathbb{S}_2$, i.e. we have that $x >_{\mathbb{S}_2} y$, $x, y \in \mathbb{S}_2$ is equivalent to $x - y >_{\mathbb{S}_2} 0$. With the notation of Definition 3, $x \in \mathbb{S}_2 = 0$ is equivalent to $x = a + bz_1 + cz_2$ and $a = b = c = 0$.

To decide whether $x \in \mathbb{S}_2$ is positive or negative we first consider small values of z_1, z_2, basically we look aroung $z_1 = z_2 = 0$. Notice that, z_1, z_2 are dropping times and should be assumed positive. However, in Section 1 we discuss the cases where $z_1 \leq 0$ and/or $z_2 \leq 0$ as well. These cases are irrelevant to the situation where z_1, z_2 are assumed small, they must be positive. These cases are going to be useful when we introduce offsets in Section 2. Indeed, in this the dropping times are always positive but given by expressions of the form $a + b(Offset_1 + Offset_{12}z_2 + z_1) + c(Offset_2 + z_2)$ for instance, see Formulas (7) and (8) so the values of z_1, z_2 may be negative.

How to Assert Positivity in $\mathbb{S}_2$ if z_1, z_2 are Small

To assert that a member $x \in \mathbb{S}_2$, $x = a + bz_1 + cz_2$, is positive or negative we need to consider various cases.

Case 1: $a \neq 0$**:** In this case the sign of a determines whether $x > 0$ or $x < 0$. This is due to the fact that by assuming the values of z_1, z_2 small enough (equal 0) the value of x can be turned the same sign as the one of a. Let us assume $a > 0$ to fix ideas. The value of x is positive but only for $z_1, z_2 = 0$. To enforce $x > 0$ the inequality

$$a > -bz_1 - cz_2$$

must be true. This may be done by computing bounds for z_1, z_2

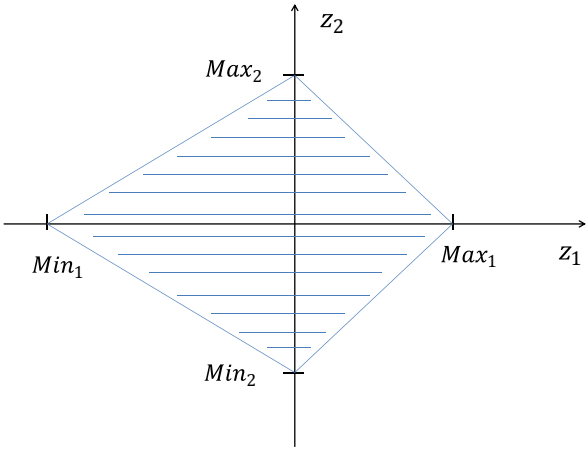

Fig. 8 The convex region defined by the computed bounds of z_1, z_2 when $a > 0$ or $a < 0$

$$z_1 > -\frac{a}{b} \text{ if } b > 0,$$
$$z_1 < -\frac{a}{b} \text{ if } b < 0,$$
$$z_2 > -\frac{a}{c} \text{ if } c > 0,$$
$$z_2 < -\frac{a}{c} \text{ if } c < 0,$$

each time a member $x \in \mathbb{S}_2$ is tested for positivity, the smallest of the upper bounds and the highest of the lower bounds are computed. At the end of the program execution the output is correct for dropping times with values $Min_1 < z_1 < Max_1$ and $Min_2 < z_2 < Max_2$, and for values of (z_1, z_2) that belong to the convex region determined by the bounds, see Fig. 8. Indeed, if the program ends with bounds

$$\begin{aligned} Min_1 < z_1 < Max_1, \\ Min_2 < z_2 < Max_2, \end{aligned} \tag{3}$$

for each couple $(z_1, z_2) = \alpha_1(Min_1, 0) + \alpha_2(Max_1, 0) + \alpha_3(0, Min_2) + \alpha_4(0, Max_2)$ with $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$, $\alpha_i \geq 0$ all the inequalities $x > 0$ encountered by the program are still valid. Indeed, if $x = a + bz_1 + cz_2 > 0$ for z_1, z_2 satisfying (3), it holds:

$$\begin{aligned} &a + b(\alpha_1 Min_1 + \alpha_2 Max_1) + c(\alpha_3 Min_2 + \alpha_4 Max_2) = \\ &\alpha_1 \underbrace{(a + bMin_1)}_{>0} + \alpha_2 \underbrace{(a + bMax_1)}_{>0} + \alpha_3 \underbrace{(a + cMin_2)}_{>0} + \alpha_4 \underbrace{(a + cMax_2)}_{>0} > 0. \end{aligned}$$

Case 2: $a = 0$ **and** $z_1 \geq 0$**:** If $a = 0$ we have to find the conditions ensuring that

$$x = bz_1 + cz_2 > 0 \text{ or } x = bz_1 + cz_2 < 0.$$

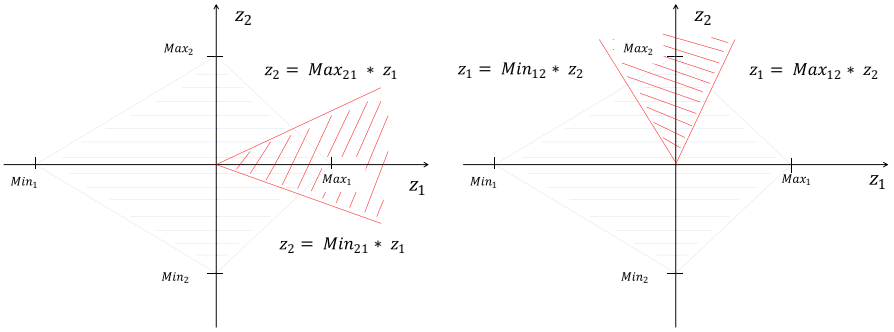

Fig. 9 The convex regions defined by the computed bounds of z_1, z_2 when $a = 0$. On the Left $z_2 \leq z_1$ $z_1 \geq 0$. On the right $z_1 \leq z_2$, $z_2 \geq 0$

If $b < 0$ we conclude that $x < 0$ because this is true for $z_1, z_2 = 0$ and $z_1 > 0$. However, in order to ensure $x < 0$ the value of z_2 must be bounded,

$$\text{if } c < 0,\ z_2 \text{ must satisfy } z_2 > \underbrace{-\frac{b}{c}}_{<0} z_1,$$

$$\text{if } c > 0,\ z_2 \text{ must satisfy } z_2 < \underbrace{-\frac{b}{c}}_{>0} z_1.$$

In conclusion, each time the program must decide whether a member $x \in \mathbb{S}_2$ with $a = 0$ is positive or negative, with $z_1 \geq 0$, the answer depends on the sign of b and the bounds Max_{21} and Min_{21} are updated by

$$Max_{21} = min\big(-\frac{b}{c}, Max_{21}\big) \text{ if } c < 0,$$
$$Min_{21} = max\big(-\frac{b}{c}, Min_{21}\big) \text{ if } c < 0.$$

At the end the values of z_1 and z_2 are related by

$$\underbrace{Min_{21}}_{<0} z_1 \leq z_2 \leq \underbrace{Max_{21}}_{>0} z_1, \tag{4}$$

the region that is defined by these inequalities and which corresponds to the region where the computations done by the program are correct is shown in the left of Fig. 9.

Similarly, if $b > 0$ we conclude that $x > 0$ because this is true for z_1, z_2 small enough,$z_1 \geq 0$, and z_2 has still to be bounded to ensure that $x > 0$. The bounds are

$$\text{if } c < 0,\ z_2 \text{ must satisfy } z_2 < \underbrace{-\frac{b}{c}}_{<0} z_1,$$

$$\text{if } c > 0,\ z_2 \text{ must satisfy } z_2 > \underbrace{-\frac{b}{c}}_{>0} z_1.$$

The bounds Max_{21} and Min_{21} are uptaded along the computations to ensure that at the end of the program (4) is required to validate the computations.

Case 3: $a = 0$ **and** $z_2 \geq 0$**:** This case is similar to the case 2 with the roles played par z_1 and z_2 inverted. It leads to the definition of new bounds Min_{12}, Max_{12} such that

$$\underbrace{Min_{12}}_{<0} z_2 \leq z_1 \leq \underbrace{Max_{12}}_{>0} z_2,$$

the region that is defined by these inequalities and which corresponds to the region where the computations done by the program are correct is shown in the right of Fig. 9.

Case 4: $a = 0$ **and** $z_2 \leq 0$**:** We have to find the conditions ensuring that

$$x = bz_1 + cz_2 > 0 \text{ or } x = bz_1 + cz_2 < 0.$$

Hence, if $c > 0$ we conclude that $x < 0$ because this is true for $z_1, z_2 = 0$ and $z_2 < 0$. However, in order to ensure $x < 0$ the value of z_1 must be bounded,

$$\text{if } b < 0,\ z_1 \text{ must satisfy } z_1 > \underbrace{-\frac{c}{b}}_{>0} z_2,$$

$$\text{if } b > 0,\ z_1 \text{ must satisfy } z_1 < \underbrace{-\frac{c}{b}}_{<0} z_1.$$

If $c < 0$ we conclude that $x > 0$ and

$$\text{if } b < 0,\ z_1 \text{ must satisfy } z_1 < \underbrace{-\frac{c}{b}}_{>0} z_2,$$

$$\text{if } b > 0,\ z_1 \text{ must satisfy } z_1 > \underbrace{-\frac{c}{b}}_{<0} z_1.$$

Each time the program must decide whether a member $x \in \mathbb{S}_2$ with $a = 0$ is positive or negative, $z_2 \leq 0$, the answer depends on the sign of c and the bounds Max_{12} and Min_{12} are updated by

$$Max_{12} = max\big(-\frac{c}{b}, Max_{12}\big) \text{ if } b < 0,$$
$$Min_{12} = min\big(-\frac{b}{c}, Min_{12}\big) \text{ if } b > 0.$$

At the end the values of z_1 and z_2 are related by

$$\underbrace{Min_{12}}_{>0} z_2 \le z_1 \le \underbrace{Max_{12}}_{<0} z_2, \tag{5}$$

the region that is defined by these inequalities and which corresponds to the region where the computations done by the program are correct is shown in the left of Fig. 10.

Case 5: $a = 0$ **and** $z_1 \le 0$**:** This case is similar to the case 4 with the roles played par z_1 and z_2 inverted. It leads to the definition of new bounds Min_{21}, Max_{21} such that (4) is valid, see the right of Fig. 10.

Case 6: $a = 0, b = 0$**:** In this case the positivity of $x = cz_2$ depends on the signs of c and z_2.

Case 7: $a = 0, c = 0$**:** In this case the positivity of $x = bz_1$ depends on the signs of b and z_1.

How to Assert Positivity in $\mathbb{S}_2$ for Any z_1, z_2.

Given a member $x \in \mathbb{S}_2$, conditions are stated in Section 1 that ensure $x > 0$ or $x < 0$. The conditions are not complete even is z_1, z_2 are assumed small. Indeed, the union of the hatched regions around the origin in Figs. 9 and 10 do not necessarily cover the region close to the origin.

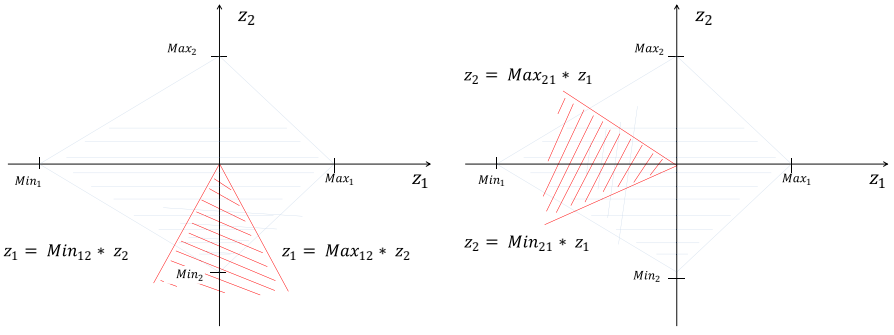

Fig. 10 The convex regions defined by the computed bounds of z_1, z_2 when $a = 0$. On the Left $z_2 \ge z_1$ $z_2 \le 0$. On the right $z_1 \le z_2$, $z_2 \ge 0$

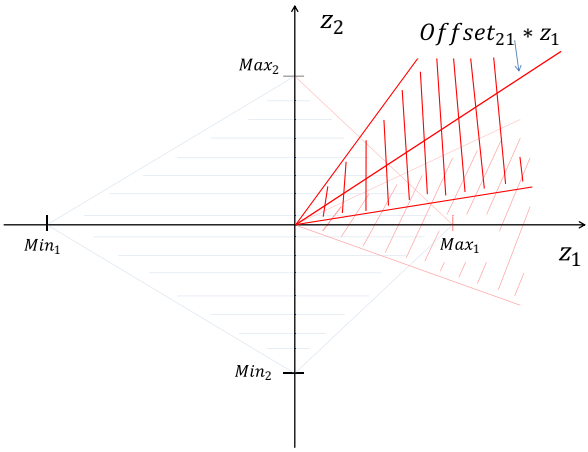

Fig. 11 Illustration of the region covered by the computations when the program is run with an offset $Offset_{21}$ ensuring that the values computed are valid in a region $Offset_{21}z_1 + Min_{21}z_1 \leq z_2 \leq Offset_{21}z_1 + Max_{21}z_1$

For instance, if the program is run with the aim of covering the values of z_1, z_2 small in the $z_1 > 0, z_2 > 0$ region. The program has to be executed to cover the cases 1, 2 and 3 of Section 1. However, it may happen that $Max_{21} < 1$ and $Max_{12} < 1$ at the end of the computations and the region $z_2 > Max_{21}z_1$ and $z_1 > Max_{12}z_2$ $(z_1, z_2 > 0)$ is not covered by the computations.

In this case we introduce offsets to sweep the region covered by the computations, see Fig. 11,

$$\begin{aligned} z_1' &= Offset_{12}z_2 + z_1, \\ z_2' &= Offset_{21}z_1 + z_2. \end{aligned} \tag{6}$$

The $x \in \mathbb{S}_2$ with offset are of the form $x = a + bz_1 + c\big(Offset_{21}z_1 + z_2\big)$ and $x = a + b\big(Offset_{12}z_2 + z_1\big) + cz_2$. By running the program with various values of the offsets, we sweep the region covered by the computations around the origin, see Fig. 11.

As a first conclusion, the program must be started to compute the rendezvous value for the dropping times z_1, z_2 small. The program must be executed to cover the cases of Section 1. Then, the program has to be run with offsets (6) in order to cover a region around the origin.

To finish, the optimal strategy pairs may be around another point in the (z_1, z_2) plane than the origin. In [3] we identify that the optimal strategy pairs are located around (0, 0) and (*D*/2, *D*/2)[5]. To cover the case $z_1 = D/2, z_2 = D/2$ we need to introduce another offsets, $Offset_1$ and $Offset_2$ (both equal to *D*/4 in this particular situation). Generally, the member of $\mathbb{S}_2$ are then represented by expressions of the form

[5] There are as well solutions where only one marker is used, when $z_1 = D/4$ z_2 is not used and vice-versa. These cases are covered with the computations for only one marker.

$$x = a + b\Big(\underbrace{Offset_1 + z_1}_{z_1'}\Big) + c\Big(\underbrace{Offset_2 + Offset_{21}z_1 + z_2}_{z_2'}\Big), \tag{7}$$

and

$$x = a + b\Big(\underbrace{Offset_1 + Offset_{12}z_2 + z_1}_{z_1'}\Big) + c\Big(\underbrace{Offset_2 + z_2}_{z_2'}\Big), \tag{8}$$

The expressions denoted z_1' and z_2' in (7) and (8) are the effective dropping times, z_1 and z_2 the indeterminates assumed small and satisfying bounds like in (3), (4), (5). $Offset_1, Offset_2, Offset_{12}, Offset_{21}$ are parameters of the program. The program output expressions for the rendezvous times that are member of $\mathbb{S}_2$, in the form of (7) or (8) and bounds $Min_1, Min_2, Max_1, Max_2$ see Fig. 8 and bounds $Min_{12}, Max_{12}, Min_{21}, Max_{21}$, see Figs. 9 and 10. The bounds determine the region in which the computations of the rendezvous times, as expressions of $\mathbb{S}_2$ are valid.

With these computations we can prove the following Theorem 2.

Theorem 2 (Two Marker Games) *The optimal solutions for the rendezvous problem on the line with two markers* (*one for each player*), *with initial distance between the players D, are given by, see Fig.* 12,

$$(0; FFBB - 0; FFBB)$$
$$(D/4; FBBFF - D/4; FFBBB)$$
$$(D/2; FBF - D/2; FBB)$$

The times at which directions can change are

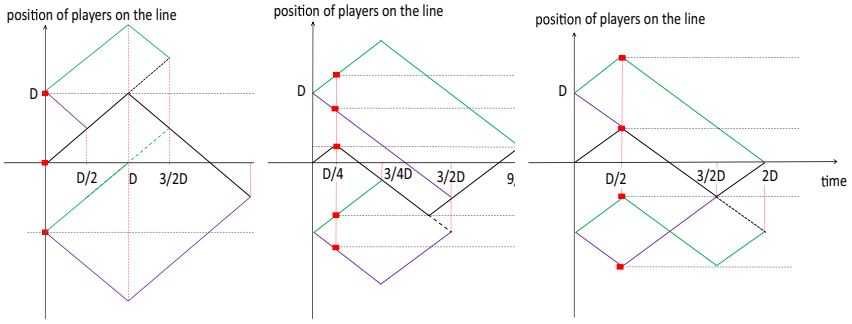

Fig. 12 Solutions of the two markers game. From left to right the dropping times are (0, 0), (D/4, D/4), (D/2, D/2). The left solution is already known, see [22]

$$\begin{array}{c} 0,\ D/2,\ D, \\ D/4,\ 3D/4,\ 5D/4,\ 3D/2, \\ D/2,\ 3D/2, \end{array} \tag{9}$$

and the corresponding rendezvous times are

$$\begin{aligned} &t^1 = 3D/2,\ t^2 = D/2,\ t^3 = 3D/2,\ t^4 = 5D/2, \\ &t^1 = 9D/4,\ t^2 = 3D/2,\ t^3 = 3D/4,\ t^4 = 3D/2, \\ &t^1 = 2D,\ t^2 = D/2,\ t^3 = 2D,\ t^4 = 2D. \end{aligned}$$

The rendezvous Value R_2 (the average of the rendezvous times) for all these solutions is

$$R_2 = 3D/2.$$

Notice that the strategy pair provided in Theorem 1 is an optimal strategy pair for the two markers game as well, the rendezvous Values being the same, i.e. two markers instead of one do not help.

The strategy pair where the dropping times are $z_1 = z_2 = 0$ is proved optimal in [22].

Proof We know from [3] that with the intial distance $D = 1$ optimal solutions are for $z_1, z_2 \in [0, 0.00016]$ or $z_1, z_2 \in [0.5 - 0.00016, 0.5 + 0.00016]$. The solution for $z_1 = 1/4$ and z_2 any value uses only one marker and is covered by Theorem 1.

We start the computations for z_1, z_2 small and positive, i.e. around the origin, and $z_1 \geq 0$. The first execution leads to an optimal rendezvous value of 3/2 for $z_1 = z_2 = 0$ and the results are correct for

$$\begin{aligned} z_2 &\leq \frac{1}{6} z1. \\ z_2 &\geq -\frac{1}{4} z1, \end{aligned}$$

this means that $Max_{21} = 1/6$ and $Min_{21} = -1/4$, see Fig. 9. The second inequality is not active since $z_2 \geq 0$. We run again the program with $Offset_{21} = 1/6$ in order to sweep the valid region, see Fig. 11 and again we obtain an optimal rendezvous value of 3/2 for $z_1 = z_2 = 0$. However, now the results are correct for

$$\begin{aligned} z_2 &\leq \left(\frac{1}{6} + \frac{1}{30}\right) z1 = \frac{1}{5} z_1, \\ z_2 &\geq \left(\frac{1}{6} - \frac{1}{6}\right) z1 = 0. \end{aligned}$$

Up to now the best rendezvous value found is 3/2 and this is valid for $z_2 \leq z_1/5$. We continue the computations until $Max_{21} \geq 1$ and check that the optimal rendezvous value is still 3/2 for $z_1 = z_2 = 0$.

Then, we restart the computations but assuming now that $z_2 \geq 0$ in order to cover the entire region $z_1 \geq 0, z_2 \geq 0$. Because z_1 and z_2 are switched and the problem symmetric we obtain the same results than for $z_2 \leq z_1$.

To conclude, we still have to consider the bounds for z_1, z_2 obtained by comparisons of the case 1 of Section 1 ($a \neq 0$) the smallest bound encountered show that $z_1 \geq 0.083$ and $z_2 \geq 0.035$ which cover the region $z_1, z_2 \in [0, 0.00016]$ of [3]. This shows that the optimal rendezvous value is 3/2 obtained for $z_1 = z_2 = 0$.

Similar computations around $z_1 = z_2 = 0.5$ shows that the results are correct.

Acknowledgements The authors are indebted to the reviewers and Editor in Chief for the precise and useful comments.

Funding Open access funding provided by University of Geneva.

Data Availability Statement This manuscript has no associated data.

Declarations

Conflicts of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. King JC (1976) Symbolic execution and program testing. Commun ACM 19(7):385–394
2. Alpern S, Gal S (1995) Rendezvous search on the line with distinguishable players. SIAM J Control Optim 33(4):1270–1276
3. Leone P, Alpern S (2018) Rendezvous search with markers that can be dropped at chosen times. Naval Research Logistics (NRL) 65(6–7):449–461
4. Alpern S (1976) Hide and Seek Games. Institut fur Hohere Studien, Wien
5. Alpern S (2002a) Rendezvous search: A personal perspective. Oper Res 50(5):772–795. https://doi.org/10.1287/opre.50.5.772.363
6. Alpern S, Gal S (2006) The Theory of Search Games and Rendezvous. International Series in Oper Res Manag Sci, Springer US, https://books.google.ch/books?id=BBrvBwAAQBAJ
7. Alpern S, Fokkink R, Gasieniec L, Lindelauf R, Subrahmanian V (2013) Search theory. Springer
8. Francis BA, Maggiore M (2016) Flocking and rendezvous in distributed robotics. Springer
9. Gu Z, Wang Y, Hua QS, Lau F (2017) Rendezvous in Distributed Systems. Springer
10. Kranakis E, Krizanc D, Markou E (2010) The mobile agent rendezvous problem in the ring. Synthesis lectures on distributed computing theory 1(1):1–122
11. Anderson EJ, Weber RR (1990) The rendezvous problem on discrete locations. J Appl Prob 27(4):839–851. http://www.jstor.org/stable/3214827

12. Weber R (2012) Optimal symmetric rendezvous search on three locations. Math Oper Res 37(1):111–122. https://doi.org/10.1287/moor.1110.0528
13. Alpern S (1995) The rendezvous search problem. SIAM J Control Optim 33(3):673–683. https://doi.org/10.1137/S0363012993249195
14. Anderson EJ, Essegaier S (1995) Rendezvous search on the line with indistinguishable players. SIAM J Control Optim 33(6):1637–1642
15. Baston V (1999) Note: Two rendezvous search problems on the line. Naval Research Logistics (NRL) 46(3):335–340
16. Gal S (1999) Rendezvous search on the line. Oper Res 47(6):974–976. https://doi.org/10.1287/opre.47.6.974
17. Han Q, Du D, Vera J, Zuluaga LF (2008) Improved bounds for the symmetric rendezvous value on the line. Oper Res 56(3):772–782. https://doi.org/10.1287/opre.1070.0439
18. Alpern S, Beck A (1999) Rendezvous search on the line with limited resources: Maximizing the probability of meeting. Oper Res 47(6):849–861. https://doi.org/10.1287/opre.47.6.849
19. Alpern S, Beck A (2000) Pure strategy asymmetric rendezvous on the line with an unknown initial distance. Oper Res 48(3):498–501. https://doi.org/10.1287/opre.48.3.498.12432
20. Baston V, Gal S (1998) Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. SIAM J Control Optim 36(6):1880–1889. https://doi.org/10.1137/S0363012996314130
21. Ozsoyeller D, Beveridge A, Isler V (2013) Symmetric rendezvous search on the line with an unknown initial distance. IEEE Trans Rob 29(6):1366–1379. https://doi.org/10.1109/TRO.2013.2272252
22. Baston V, Gal S (2001) Rendezvous search when marks are left at the starting points. Naval Research Logistics (NRL) 48(8):722–731
23. Howard J (1999) Rendezvous search on the interval and the circle. Oper Res 47(4):550–558. https://doi.org/10.1287/opre.47.4.550
24. Kranakis E, Santoro N, Sawchuk C, Krizanc D (2003) Mobile agent rendezvous in a ring. In: 23rd International Conference on Distributed Computing Systems, 2003. Proceedings., IEEE, pp 592–599
25. Alpern S (2002b) Rendezvous search on labeled networks. Naval Research Logistics (NRL) 49(3):256–274. https://doi.org/10.1002/nav.10011
26. Anderson EJ, Fekete SP (2001) Two dimensional rendezvous search. Oper Res 49(1):107–118. https://doi.org/10.1287/opre.49.1.107.11191
27. Chester EJ, Tütüncü RH (2004) Rendezvous search on the labeled line. Oper Res 52(2):330–334. https://doi.org/10.1287/opre.1030.0085
28. Kikuta K, Ruckle WH (2007) Rendezvous search on a star graph with examination costs. Eur J Oper Res 181(1):298–304. https://doi.org/10.1016/j.ejor.2006.06.017
29. Chang CS, Liao W, Lien CM (2015) On the multichannel rendezvous problem: Fundamental limits, optimal hopping sequences, and bounded time-to-rendezvous. Math Oper Res 40(1):1–23. https://doi.org/10.1287/moor.2014.0680
30. Czyzowicz J, Kranakis E, Krizanc D, Narayanan L, Opatrny J (2019) Search on a line with faulty robots. Distrib Comput 32(6):493–504
31. Czyzowicz J, Killick R, Kranakis E (2018) Linear rendezvous with asymmetric clocks. In: 22nd International Conference on Principles of Distributed Systems (OPODIS 2018), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik
32. Bampas E, Czyzowicz J, Gąsieniec L, Ilcinkas D, Klasing R, Kociumaka T, Pająk D (2019) Linear search by a pair of distinct-speed robots. Algorithmica 81(1):317–342
33. Chrobak M, Gąsieniec L, Gorry T, Martin R (2015) Group search on the line. In: International Conference on Current Trends in Theory and Practice of Informatics, Springer, pp 164–176
34. Czyzowicz J, Killick R, Kranakis E, Krizanc D, Narayanan L, Opatrny J, Pankratov D, Shende S (2021) Group evacuation on a line by agents with different communication abilities. https://arxiv.org/abs/2109.12676
35. Fomin FV, Thilikos DM (2008) An annotated bibliography on guaranteed graph searching. Theoret Comput Sci 399(3):236–245

36. Hohzaki R (2016) Search games: Literature and survey. J Oper Res Soc Japan 59(1):1–34
37. Leone P, Cohen N (2021) Rendezvous on the line with different speeds and markers that can be dropped at chosen time. https://arxiv.org/abs/2109.00905
38. Leone P, Buwaya J, Alpern S (2021) Search-and-rescue rendezvous. Euro J Oper Res
39. Bradley AR, Manna Z (2007) The calculus of computation: decision procedures with applications to verification. Springer Science & Business Media
40. Kroening D, Strichman O (2016) Decision procedures. Springer