# Introduction to AI - This Bot is on Fire

**Dharm Maheshkumar Patel**
**dp1482@scarletmail.rutgers.edu**

**Pradhyumna Kiledar**
**pk811@scarletmail.rutgers.edu**

## Abstract

In our project "The Bot is on Fire," we aim to build effective search algorithms in an attempt to cope with spaces where two main uncertainties exist: the space topology and the spread of fire. The goal is to build a smart and strong bot which moves around an arbitrarily designed grid-based world. This environment has a probabilistic adversary, the fire that dynamically closes potential paths to the goal throughout the game. Throughout the project, we examine various search algorithms and implement optimizations to solve this challenging problem efficiently.

## 1) Environment and Ship Setup:

The environment is a 40x40 square maze, represented as a 2D array of 40x40 ignitable cell objects. Here, white cells are open cells, and black cells are blocked or closed cells. To create the maze, we start by opening a random single inner blocked cell. Then we continue to open those cells which are still closed but have one open neighbor, i.e., frontier cells, repeatedly. This creates a maze-like structure. Furthermore, half of the dead-end cells—open cells with only one open neighbor—are opened to expand the playable area without diminishing much of the maze's complexity.
During initialization, robot, fire, and buttons are assigned to three random open cells. During each tick, the robot can go either right, left, up, or down. At the same time, the fire spreads to the neighboring cells with probability determined by the following formula:
$1 - (1-q)^k$

Here, k is the number of adjacent burn units, and q is the flammability parameter, which can be any value between 0 and 1. The flammability parameter determines the rate of the fire spread—greater values result in more rapid and intense fire spread.

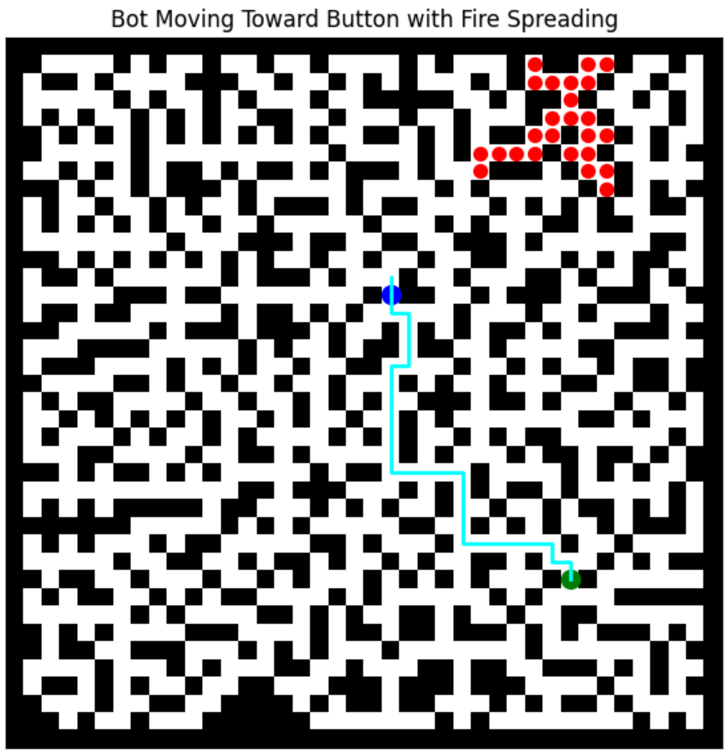
## 2) Bot Strategies:

### Bot 1:

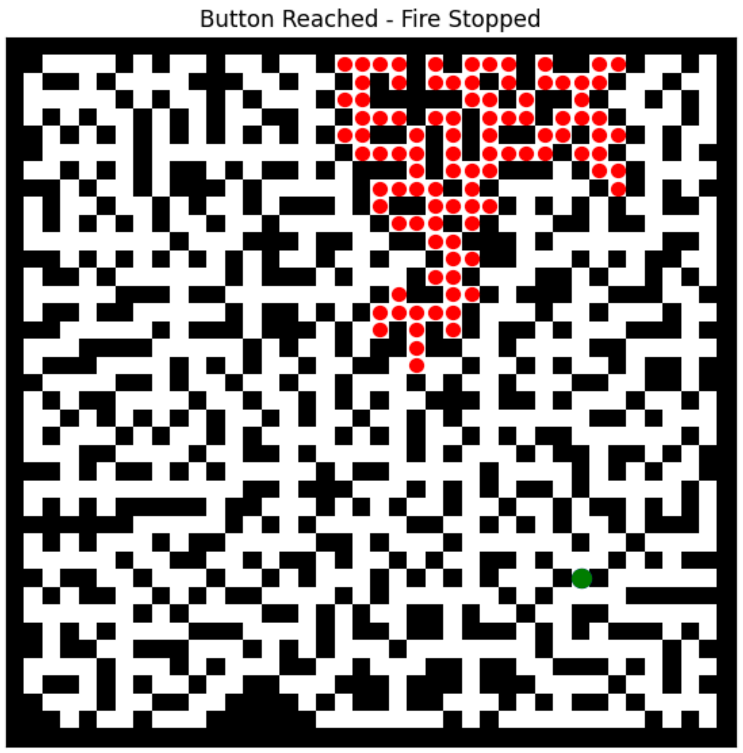
Bot 1 will apply the BFS method so that it can find the shortest route to the button. The route is calculated while avoiding the first fire cell, and then the bot moves along the marked route.

There is no arrangement made by this bot for the advancing burning while moving. It is using the BFS algorithm since it is concerned about reaching the button in the minimum number of moves, considering speed and not spreading of the fire dynamically.

Here the two following figures describe the visual display of bot 1.



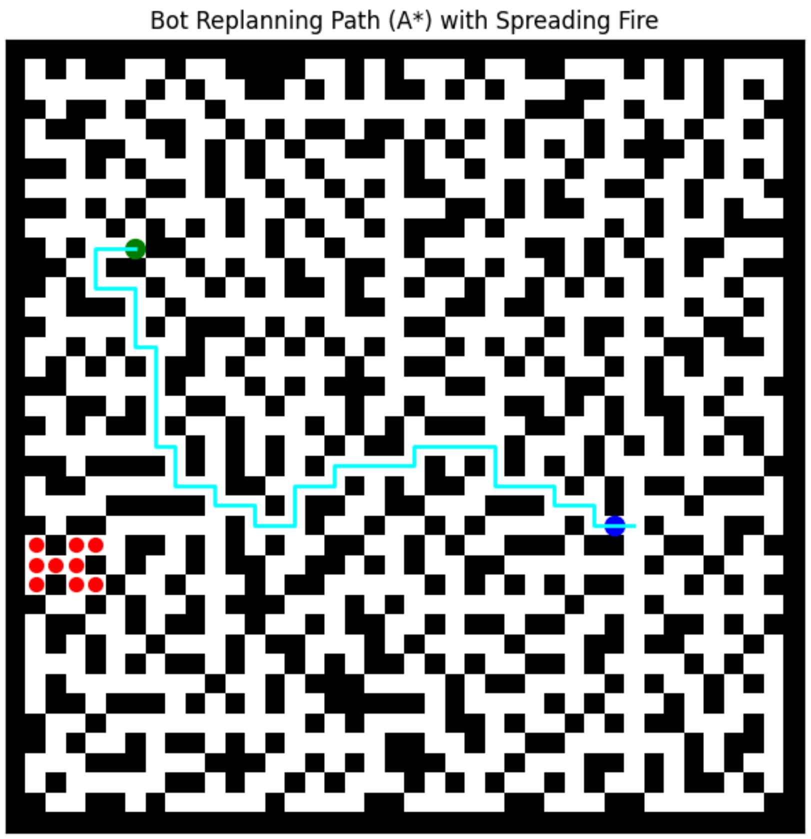
**Bot 1 simulation**



**Bot 1 simulation (End position)**

## Bot 2:

Bot 2 operates by always going in the shortest distance towards the button with each step and changing its direction as the fire spreads. It avoids the fire by wholly redefining its trajectory every time in an attempt to remain undamaged. Bot 2 achieves this with the application of the A-star (A*) algorithm, which will select the best route via a minimum duration based on the distance to the button and assumed remaining distance. Bot 2 is thus clever and robust in order to avoid spreading of the fire. Here the following is the graphical representation of bot 2.

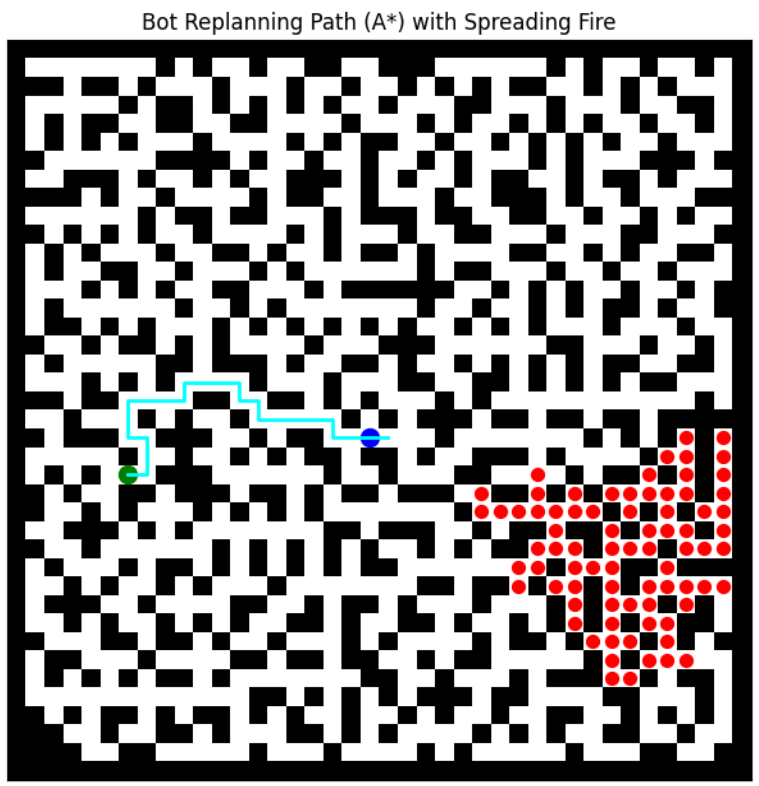


**Bot 2 simulation**

## Bot 3:

Bot 3 moves through the maze by finding the shortest path to the button step by step. Not only the fire cells in its present step, but also any bordering fire cells are skipped to have a safe path. If there is no such path, it uses the shortest path with only the present fire cells being skipped. For this, Bot 3 utilizes the A-star (A*) algorithm, specifically designed to find optimal paths within a dynamic environment. The A* algorithm utilizes a heuristic function, e.g., Manhattan distance, for estimating the minimum bot-to-button distance. This helps the bot place highest priority upon short and safe paths.

The algorithm flags unsafe cells, including fire cells and adjacent cells, and does not utilize them in planning the path. It goes through the lowest-cost paths first using a priority queue, taking into account adjacent cells irrespective of walls, fire cells, and adjacent unsafe cells. If a valid path exists, it reconstructs and returns the list of moves; otherwise, it returns an empty list. This approach renders Bot 3 highly adaptive and able to react to the spread of the fire well with safety being its top priority. By dynamically adapting its path based on the spread of the fire, Bot 3 achieves a balance between reaching the button in the shortest time and remaining safe. The graphical depiction of bot 3 is shown in the figure below.



**Bot 3 simulation**

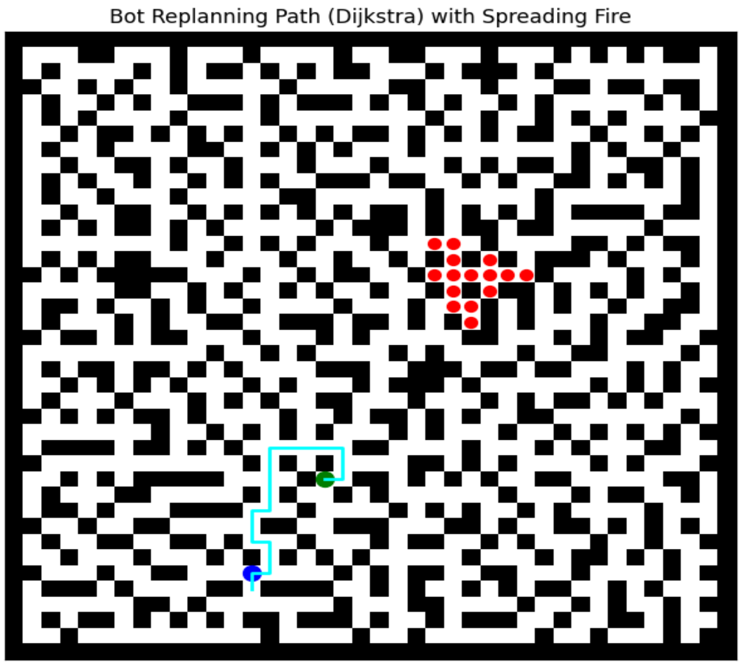
## Bot 4:

Bot 4 uses a smart method called fire-aware predictive navigation to navigate the maze without being injured by the fire. It does not simply avoid the fire but actually computes the risk of each cell based on how close it is to the fire and how likely the fire will propagate there in the near future. To predict where the fire will spread, Bot 4 in advance simulates the spread of the fire for the next couple of steps. The steps ahead that it simulates is a function of the flammability parameter q, which indicates how fast the fire spreads. For small q, the fire spreads slowly and Bot 4 can simulate steps further ahead accurately. When q is big, the fire spreads quickly and looking too far ahead can lead to overestimation of danger, freezing the bot, or making an incorrect move. Therefore, for big values of q, Bot 4 looks fewer steps ahead in order not to overestimate and to move smoothly.

We initially experimented with the formula **predict_steps = round(1/q)** to determine the number of steps ahead. There was a weakness in the formula. For example, when $q = 0.1$, it overestimated 10 steps ahead, and For $0.7<=q<0.9$, the equation always forecasted 1 step, which is less than the actual for $q<0.8$. So, for the range $0.3<=q<0.8$, the equation was underestimating the fire spread. For example, when $q = 0.4$, it forecasted 2 steps, which was insufficient to model the fire spread, and hence poor decisions were made. After having executed and compared simulations for a variety of values of q, we established the following optimal ranges for predicting steps in advance:

If $q < 0.1$, then it predicts 6 steps ahead.
If $q < 0.3$, then it predicts 4 steps ahead.
If $q < 0.6$, then it predicts 3 steps ahead.
If $q < 0.8$, then it predicts 2 steps ahead.
Else, then it predicts 1 step ahead.

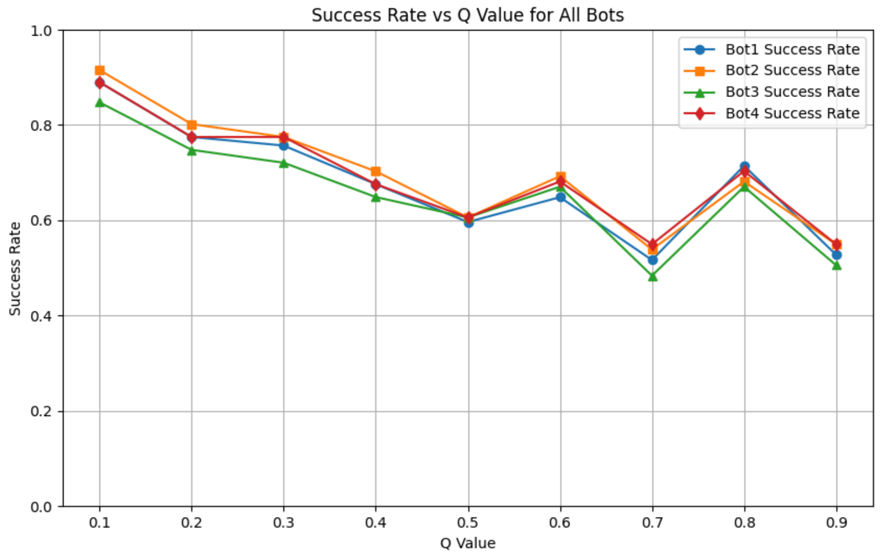
These figures were chosen because they trade off against each other the utility of being able to predict fire spread and keeping the bot as responsive and quick as possible. Within these limits, Bot 4 can successfully chart a course that will steer clear of dangerous areas even before the fire has reached the location. If the intended route is found to be too dangerous once the fire continues further, Bot 4 quickly re-analyzes and maps out a less dangerous path.



**Bot 4 simulation**

To implement this strategy, Bot 4 uses a modification of Dijkstra. The modification uses a risk function that considers both the current fire and where the fire is most likely to spread next. Bot 4 simulates the fire spread before it plans a path so that it can ahead of time search for hazardous areas. It uses a weighted graph where the cost of moving through a cell is determined by both its hazard level and distance. Bot 4 regularly updates its path with the latest fire prediction and is both flexible and safe, and it is very good at solving the maze without being caught by the fire.

## 3) Test environments and evaluate the performance of the bot:



**Q values vs Success Graph**

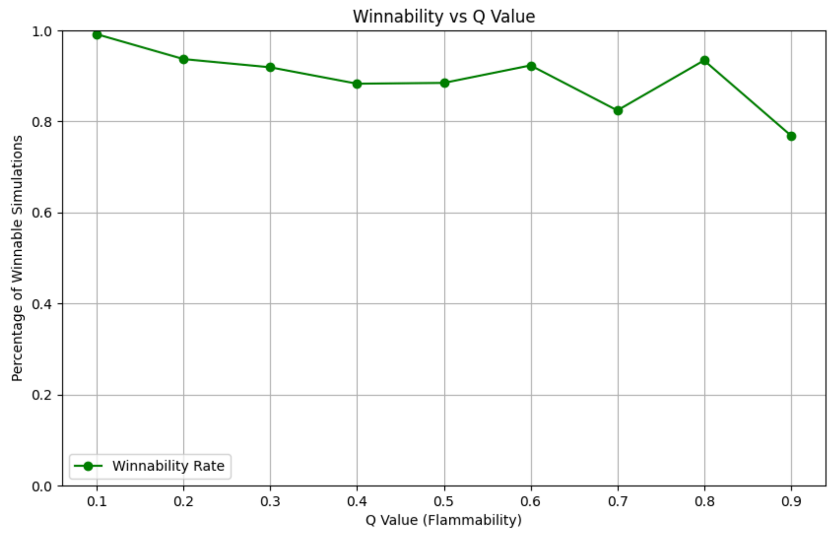
The graph "Success Rate vs Q Value for All Bots" demonstrates the performance of four bots (Bot 1, Bot 2, Bot 3, and Bot 4) when tested with varying values of the flammability parameter q. The success rate of all bots is plotted versus q ranging from 0.1 to 0.9. The graph shows the way the success rate of the bot is a function of fire spread rate and how q is closer to 0 when the fire spread rate is low and closer to 1 when the fire spread rate is high.

For an equivalent fair comparison, the same condition was employed for testing all four bots with the same initial position of the button, fire, and bots. Different q values were used in the simulation, and each bot's success percentage was calculated in terms of safe arrival at the button. The result was compared and graphed out.

We can observe from the graph that, as q tends to 0, all the bots have a greater success rate. This is because of the gradual spread of fire, giving the bots sufficient time to move around in the maze and reach the button without being destroyed. As q increases, the success rate of all the bots decreases, with the decrease being more for the bots that do not use dynamic fire spread.

## 4) Test to determine whether that simulation was winnable:

The Winnability Graph "Winnability vs Q Value" shows the ratio of winnable simulations for different values of the flammability parameter q from 0.1 to 0.9. The graph tells us about how the winning probability (arriving at the button) changes with a higher rate of fire spread. For q close to 0, the rate of fire spread is low, and bots tend to win. As q increases, the fire travels faster, making the probability of winning less.



**Q Value VS Percentage of winnable simulations**

For these test runs, initial positions of button, fire, and all the bots are identical. If in a test run no bot is able to reach the button, we utilize an ideal bot to check whether there is any path or not. If an ideal bot reaches the button, then the state is winnable. If the ideal bot cannot reach the button, then the state is unwinnable. In this way we are thinking about realistic bot ability as well as theoretical capability.

**Winnability Test Implementation:**

**To test for winnability, we do the following:**

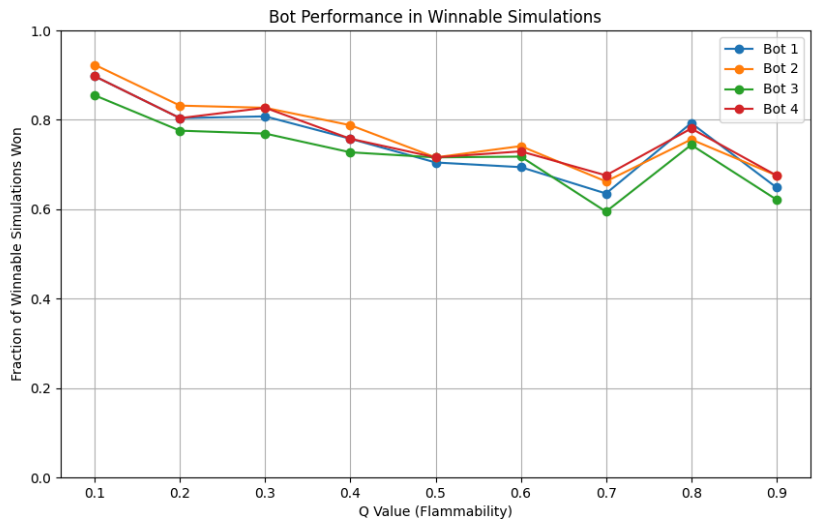
- **Direct Winnability:** If we can find a bot which can reach the button, then the simulation is winnable.
- **Theoretical Winnability:** Simulate an ideal bot with perfect knowledge of the fire spread. If the ideal bot can reach the button, the simulation is winnable. If there is no path even for the ideal bot, then the simulation is not winnable.

## Observations from the Graph:

The graph indicates that as q increases, the number of winnable simulations reduces. This is because higher values of q imply that the fire spreads more quickly, and therefore it is more difficult for the bots to access the button. But even when the q value is high, there are still winnable simulations, particularly when an optimal bot with full knowledge of the spread of fire is used.

In general, the Winnability Graph illustrates well how fire spread speed affects the simplicity of the task and, in its turn, the necessity to use smart, adaptive strategies to bring the chance of success as high as possible, especially when the fire develops very fast.

## 5) Graph showing that excludes simulations from our data that were not winnable:



**Q value vs Fraction of winnable simulations won**

The plot "Bot Performance in Winnable Simulations" shows the performance of four bots (Bot 1, Bot 2, Bot 3, and Bot 4) in winnable simulations. A simulation is winnable if either one bot or the best bot can reach the button. The plot shows the frequency (probability) of each bot winning in these winnable simulations as a function of the flammability parameter q, ranging from 0.1 to 0.9.

Here in the graph we rule out non-winnable simulations by the four bots or the optimal bot. This is in order to leave us with just the cases in which winning is theoretically possible.
The probability of success for every bot is calculated as the number of successful wins by the bot divided by the number of winnable simulations. And The graph shows which bot performs better in winnable situations, regardless of the level of difficulty that is introduced by the fire spread rate.

## 6) When a bot fails to reach the button (and success was possible), why does it fail? Was there a better decision that could have been made that would have saved the bot? Support your conclusions.

In bot 4, where we used the formula predict_steps = round(1/q) to determine the number of steps ahead. There was an error in the formula. For example, when $q = 0.1$, it overestimated 10 steps ahead, and For $0.7<=q<0.9$, the equation always estimated 1 step, which is below the actual for $q<0.8$. Thus, for the range $0.3<=q<0.8$, the equation was underestimating the spread of the fire. For example, when $q = 0.4$, it forecast 2 steps, which was insufficient to model the fire spread, and hence decisions were made on the basis of that we were drawing poor result for some value of q. Having run and compared simulation for a range of various values of q, we have set the following optimal ranges in order to forecast steps ahead:

If $q < 0.1$, then it predicts 6 steps ahead.
If $q < 0.3$, then it predicts 4 steps ahead.
If $q < 0.6$, then it predicts 3 steps ahead.
If $q < 0.8$, then it predicts 2 steps ahead.
Otherwise, then it predicts 1 step.
and utilizing optimal ranges we boosted the overall performance of the bot 4.

## 7) Speculate on how you might construct the ideal bot. What information would it use, what information would it compute, and how? Do not worry too much about runtime.

The ideal bot is the bot who already knows in advance where the fire will spread. With this foresight, it can make the best choices to survive. Which we have used to find the winnable situations. However, for a real-world scenario where the bot does not know the exact future fire locations, our ideal bot is Bot 4. Bot 4 predicts the future fire spread at each step based on the q value, which represents the probability of fire spread. Bot 4 uses these predictions to strategically avoid risky cells with high danger while still attempting to reach the goal through the shortest path available. By dynamically adapting to the potential spread of the fire, Bot 4 effectively balances safety and efficiency and is therefore the best practical solution.