Pathfinding for the navigation of visually impaired people

Kai Li Lim*

School of Electrical, Electronic and Computer Engineering,
The University of Western Australia,
35 Stirling Highway, Crawley WA 6009, Australia
Email: kaili.lim@research.uwa.edu.au
*Corresponding author

Kah Phooi Seng

School of Computing and Mathematics,
Charles Sturt University,
Locked Bag 588,
Wagga Wagga NSW 2678, Australia
Email: kseng@csu.edu.au

Lee Seng Yeong

Department of Computing and Information Systems,
Sunway University,
5 Jalan Universiti, Bandar Sunway,
47500 Petaling Jaya, Selangor, Malaysia
Email: leesengy@sunway.edu.my

Li-Minn Ang

School of Computing and Mathematics,
Charles Sturt University,
Locked Bag 588,
Wagga Wagga NSW 2678, Australia
Email: lang@csu.edu.au

Sue Inn Ch'ng

Department of Computing and Information Systems,
Sunway University,
5 Jalan Universiti, Bandar Sunway,
47500 Petaling Jaya, Selangor, Malaysia
Email: sueinnc@sunway.edu.my

Abstract: A navigation system using an Android mobile device for the visually impaired is explored in this paper. This paper focuses on pathfinding algorithms and their implementations on Java platform. The boundary iterative-deepening depth-first search (BIDDFS) pathfinding algorithm is extended for bidirectional searching. Fast pathfinding is applied for the BIDDFS by reducing memory read and writes cycles, proposing the optimised BIDDFS. Fast pathfinding is also extended for the bidirectional BIDDFS, proposing the fast bidirectional BIDDFS. The fast bidirectional BIDDFS uses Java's thread feature to implement a parallel structure. The optimised BIDDFS was able to record drastic improvements in pathfinding speeds compared to the standard BIDDFS. Likewise, the fast bidirectional BIDDFS recorded significant speed improvements over the parallel bidirectional BIDDFS.

Keywords: pathfinding; visually impaired; uninformed search; bidirectional search.

Reference to this paper should be made as follows: Lim, K.L., Seng, K.P., Yeong, L.S., Ang, L-M. and Ch'ng, S.I. (2016) 'Pathfinding for the navigation of visually impaired people', *Int. J. Computational Complexity and Intelligent Algorithms*, Vol. 1, No. 1, pp.99–114.

Biographical notes: Kai Li Lim received his BEng degree (with first-class honours) in Electronic and Computer Engineering from the University of Nottingham in 2012, and the MSc degree (with distinction) in Computer Science from Lancaster University and Sunway University in 2014. He is currently a PhD candidate at the University of Western Australia with the School of Electrical, Electronic and Computer Engineering. His research interests cover the fields of navigational algorithms, autonomous mobile robots, and ubiquitous computing.

Kah Phooi Seng received her BEng (first class) and PhD from University of Tasmania, Australia, in 1997 and 2001, respectively. She is currently with the School of Computing and Mathematics, Charles Sturt University. She was a Professor at Sunway University before returning to Australia. Prior to joining Sunway University, she was an Associate Professor at School of Electrical and Electronic Engineering, Nottingham University. Her research interests include the fields of visual processing, multi-biometrics, artificial intelligence and wireless visual sensor networks.

Lee Seng Yeong graduated from the University of Nottingham in 2015 with a PhD in Engineering. He is currently a Lecturer at Sunway University. His topics of research include wireless sensor networks, image processing and embedded systems.

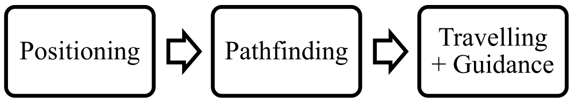
Li-Minn Ang received his BEng (first class) and PhD from Edith Cowan University, Australia, in 1996 and 2001, respectively. He is currently attached with the School of Computing and Mathematics, Charles Sturt University. His research interests include the fields of video compression, visual processing, wireless visual sensor networks and reconfigurable computing.

Sue Inn Ch'ng received her ME and PhD degrees from the University of Nottingham in 2009 and 2014, respectively. She is currently attached with the Department of Computer Science and Networked Systems, Sunway University, Malaysia. Her research interest is in biometrics, image processing and artificial intelligence.

1 Introduction

Human navigation is defined as the process for a human to travel between two points on a location, usually associated with the act of walking or running. Navigation systems are electronic devices that assist human navigation by planning and guiding the user along a route during navigation. Therefore, the process of navigation starts by identifying the position of the user (positioning); and then a route is planned for the user to take during travelling (pathfinding); and finally, the user guided by the planned path, travels to the destination (guidance). The navigation process is illustrated in Figure 1.

Figure 1 The process of human navigation



Pathfinding in navigation is defined as the plotting by a navigation system to find the best route between two points. Routes from pathfinding are normally optimised for one more of the following: shortest distance, fastest travelling time, or lowest cost. Navigation system employs pathfinding algorithms to perform pathfinding. Pathfinding algorithms are classified into informed searches and uninformed searches. In an informed search, the general direction of the destination from the current position is known. In uninformed searches, the direction of the destination is unknown from the current position. This leads to an exhaustive search from the current position until the destination is found. Uninformed searches are preferred in applications where it guarantees the shortest path from the starting point to the goal; and it has a faster node expansion speed than informed searches. Examples of uninformed search algorithms are the Dijkstra's (1959) algorithm and the iterative-deepening depth-first search (Korf, 1985). The IDDFS was proposed to solve the exponential memory requirements of the Dijkstra's algorithm with larger map size. However, the reduced memory of the IDDFS led to searching redundancy where the algorithm repeatedly expands from the starting location to find the goal.

Pathfinding algorithms are often used in video and computer games. This is due to the effective usage of artificial intelligence (AI) in video games that allows intelligent decision making and responses to produce a more dynamic and realistic environment. Examples such as Björnsson et al. (2005), Cai et al. (2011), Khantanpoka and Chinnasarn (2009), Leigh et al. (2007), Sturtevant (2012), and Wu and Zhang (2011) are few among the many. Of which, Bulitko et al. (2011) investigated real-time informed searches, Khantanpoka and Chinnasarn (2009) used informed algorithms for multi-layer searching; Leigh et al. (2007) employed an A*-like genetic algorithm for a naval real-time strategy (RTS) game; Sturtevant (2012) ran benchmarks for algorithms on a grid-based map for games; and Wu and Zhang (2011) introduced a faster A*-like algorithm with wider heuristics applied on game maps. However, there are fewer resources on pathfinding for the visually impaired (Di Giampaolo, 2010; Folmer et al., 2009; Hua et al., 2007; Oktem et al., 2008), and the available research is not as widespread. The Dijkstra's algorithm is used in Di Giampaolo (2010) and Hua et al. (2007) and the A* search is used in Folmer et al. (2009) and Oktem et al. (2008). The effort in Folmer et al. (2009) employed the visually impaired navigation on a virtual environment, translating virtual objects into text, which is read to the user in synthesised speech.

In this paper, new pathfinding algorithms are proposed for navigation of visually impaired people. The bidirectional boundary iterative-deepening depth-first search (BIDDFS) is proposed, which is an extended version of the BIDDFS. The BIDDFS (Lim et al., 2013) is an uninformed pathfinding algorithms proposed as a compromise for the Dijkstra's algorithm's exponentially increasing memory footprint on larger maps, and the IDDFS' node expansion redundancy. Experiments performed showed that the BIDDFS has a faster single-node expansion time than the iterative-deepening A* (IDA*), fringe search, and the iterative deepening depth first search (IDDFS). The bidirectional BIDDFS offers faster pathfinding by searching from both the starting and the goal until both searches meet. This effectively reduces the search area compared to unidirectional search algorithms. As a result, time and memory consumption is reduced.

The pathfinding algorithms are good for implementation on an Android device, and Android applications are programmed in Java. Java source codes can be implemented across different hardware platforms, it is known as a write once, run anywhere (WORA) language (Oracle Corporation, 2014). The pathfinding algorithms are programmed using NetBeans, an integrated development environment (IDE) by Oracle Corporation primarily for Java development. This simplifies the implementation of the pathfinding algorithms onto a navigation system running on the Android platform. Furthermore, programming the pathfinding algorithm in Java allows the identification of critical points that can be optimised for faster pathfinding. Therefore, the optimised BIDDFS is proposed in this paper. Java's thread functions allows easy implementation of parallel programming for the pathfinding algorithms, increasing the utilisation of multi-core processors found in smartphones and computers today. The parallel approach is extended for the bidirectional BIDDFS and thus the fast bidirectional BIDDFS is proposed.

The remainder of this paper is organised as follows: Section 2 proposes the bidirectional BIDDFS. Section 3 proposes a fast implementation of the BIDDFS. Section 4 describes the simulation experiments of the algorithms, comparing their pathfinding efficiency. Section 5 discusses about navigation for the visually impaired and how pathfinding can be applied. Section 6 concludes this paper.

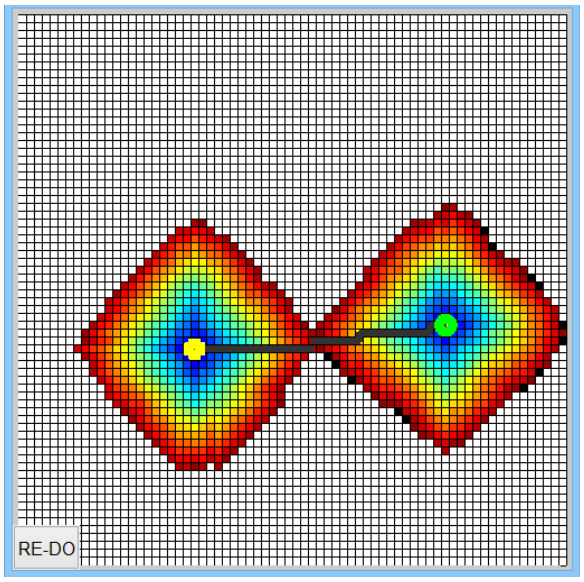
2 The proposed bidirectional BIDDFS

In this section, a new algorithm called bidirectional boundary iterative-deepening depth-first search (bidirectional BIDDFS) is proposed. The BIDDFS as previously designed by Lim et al. (2013), was enhanced to allow searching from the starting node and the goal node simultaneously, yielding higher pathfinding efficiencies. It is based on the idea that faster pathfinding will result if the search expansions of a pathfinding algorithm is to be performed from both the starting node and goal node concurrently, due to the resulting expansion area being smaller than that of a unidirectional search. Hence, bidirectional search will also lead to a reduction of memory footprint.

To index the expanded nodes from each direction (starting node direction and goal node direction), a new variable is introduced as a two-dimensional matrix with the same dimensions of the field map, with each value on the matrix representing direction, and their locations reflecting the location on the field map. This new variable is required so that the algorithm can determine whenever and wherever the two expansion directions meet, the pathfinding process will then terminate.

The location where the two expansion directions meet is called the meeting node. For each completed pathfinding process, there are two meeting nodes – each representing the direction it originates. Each meeting node will plot its way back in the opposite direction of the other using the calculated least cost path to ensure the shortest path from the meeting node to the starting or goal node. Once the route plotting is completed, there would be a path connecting the starting node to the ending node, which is the resultant route. This search pattern is as illustrated in Figure 2, with the line connecting the two starting nodes being the resultant path, with the expansions from each originating node as shown. Likewise, the bidirectional BIDDFS can be represented by the following pseudocode in Figure 3. The flowchart of the bidirectional BIDDFS is given as Figure 4 and it illustrates the logical process of the algorithm.

Figure 2 Search pattern of the bidirectional BIDDFS (see online version for colours)



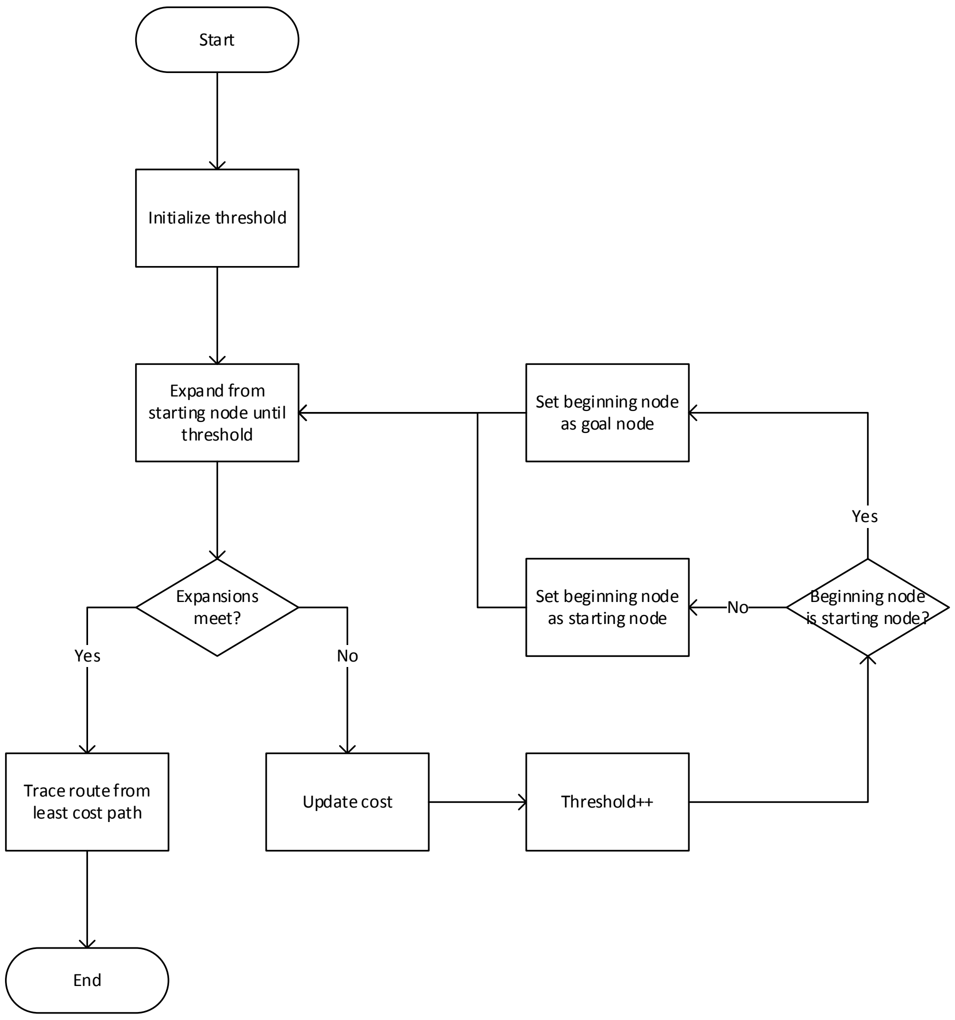
Pathfinding for navigation can be optimised for faster speeds. For example, assumptions such as BIDDFS will perform on uniform grid maps allow the cost across the map to be uniform, simplifying calculations. Calculation redundancies such as using multiple variables for calculation can allow these variables to be condensed into fewer variables. Ultimately, the BIDDFS was then optimised for faster pathfinding by reducing the memory read and write cycles. At the initialisation stage, the number of variables were reduced from six to three. Noting that some variables can be combined, i.e., having cost calculation occupying the positive values and the types of nodes occupying negative values in a matrix. This reduced the memory requirement by half. During the expansion of nodes, the reduction of variables brought one reduction of read cycle and two reductions of write cycle for every node expanded.

Figure 3 Pseudocode of the bidirectional BIDDFS

```
init
   boundary B = s
   cache C[beginning] = (0, null)
   for x in C, x != beginning
      C[x] = null
   threshold = h(start)
   reachedgoal = false
   while NOT goal=true AND B NOT empty
      fmin = ∞
      for x in F, from left to right
         (g, parent) = C[n]
         if g > threshold
            fmin = min(g, fmin)
            continue
         if x = goal
            reachedgoal = true
            break
         for s in children(n), from right to left
            g(s) = g + cost(x, s)
            if C[s] != null
               (g', parent) = C[s]
               if g(s) >= g'
                  continue
            if s in B
               remove s from B
            insert s in F past x
            C[s] = (g(s), x)
         remove n from B
      threshold = fmin
      if beginning node is starting node
         beginning node = goal node
      else
         beginning node = starting node
   if reachedgoal = true
      make path from cache
```

The open and closed sets, the cost calculation, and the initialised field, which are all condensed into a single variable ‘field’. Variables containing route directions and the node cost chart remains independent. This means that the large, closed set array is no longer present in both algorithms. When the program initialises, the ‘field’ variable is first initialised with a set of negative numbers, indicating the field map of the environment. Negative numbers distinguishes itself where it stores the state of each node, such as walls, unexpanded nodes, or the starting and goal node. Hence, the cost at each node is recorded as a positive number on the field. For the calculation of the resultant route, instead of marking down the directions to the goal node during pathfinding, directions are now calculated on the fly after pathfinding is completed. The algorithms trace back the least cost path from the goal node to the starting node and store them in a two-dimensional character array variable 'directions', which is initialised using location data of walls, goal, and starting node, obtained from 'field', which is then printed out in text. When the time saved for a single node expansion is applied across the map, pathfinding speeds can increase tremendously.

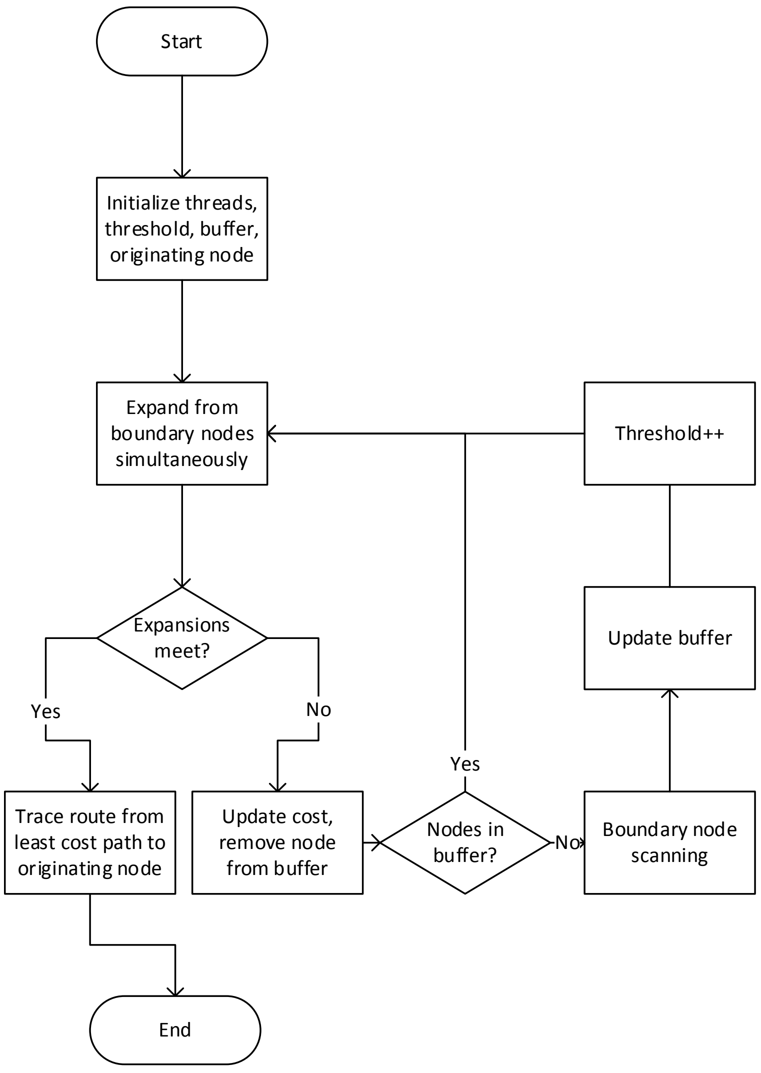
Figure 4 Flowchart of the bidirectional BIDDFS


Start
Initialize threshold
Expand from starting node until threshold
Set beginning node as goal node
Yes
Expansions meet?
Set beginning node as starting node
No
Beginning node is starting node?
Yes
No
Trace route from least cost path
Update cost
Threshold++
End

3 Fast implementation of the bidirectional BIDDFS

The optimisation of the BIDDFS has been performed in Section 2. The nature of bidirectional searches, which searches from both the starting node and the goal node, encouraged the implementation of a parallel structure. This further shortened pathfinding time. Searching in parallel means that the node expansions from both the starting and the goal nodes happen simultaneously, unlike the search that alternates between the starting node and the goal node in serial bidirectional algorithms. It is introduced by incorporating Java's thread function. The optimised BIDDFS algorithm is thus enhanced for parallel bidirectional searching, proposing the fast bidirectional BIDDFS.

Figure 5 Flowchart of the fast bidirectional BIDDFS


Start
Initialize threads, threshold, buffer, originating node
Expand from boundary nodes simultaneously
Threshold++
Update buffer
Expansions meet?
Yes
No
Yes
Trace route from least cost path to originating node
Update cost, remove node from buffer
Nodes in buffer?
No
Boundary node scanning
End

This algorithm is proposed following the analysis of the bidirectional BIDDFS, and identifying sections of the algorithms that can be simplified or omitted to fit the purpose of the application in Section 5. For example, the queue for future node expansions was simplified into a linear buffer, utilising Java's ArrayList; and that multiple variables were condensed into fewer variables. The resulting algorithm was then completely reprogrammed in Java and its pathfinding time was compared.

For parallel searching, a 'pattern' variable is instantiated to monitor the node expansions originating from the starting node and the goal node. When the expansion meets, the nodes that connects one expansion to another is called the meeting nodes. When the meeting nodes are found, the search terminates. The 'pattern' variable is placed in a dedicated thread that is introduced to detect the meeting of the two expansion patterns.

To achieve fast bidirectional searching, each pathfinding runtime from each originating node is assigned as its own class with its own thread, which will be inherited from the original pathfinding class as the parent class. A pair of buffer is used in parallel bidirectional searching for each pathfinding runtime, which prevents the clashing of node locations stored by two different runtimes. The parent class monitors the node expansion patterns for the meeting nodes, and hence the 'pattern' variable will be shared among the classes, hence allowing the parent class to terminate pathfinding when the expansion patterns meet. Therefore, there are three processes running during a parallel bidirectional search – an expansion from the starting node, an expansion from the goal node, and a process to detect the meeting of both expansions.

On the fast bidirectional BIDDFS, pathfinding from the two originating nodes are independently initialised and threaded, and then are started and run simultaneously. When the first node is expanded, boundary nodes are identified as the four-connecting nodes of the originating node. Nodes stored in the buffer are flagged for expansion and are subsequently removed from the buffer once that node has been expanded. Subsequent iterations of the algorithm will scan for the edges (boundaries) for the expanded nodes until both expansion patterns meet, and the meeting nodes are located, and the resultant route can be plotted from the least cost path. The flowchart for the fast bidirectional BIDDFS is as illustrated in Figure 5.

4 Experimental results

Pathfinding algorithms were programmed in Java and simulated using different maps and environments to measure pathfinding efficiency. Experiments are performed on open maps with no obstacles unless otherwise stated, which enables the algorithm to expand to more nodes, better demonstrating the rate of node expansion of the algorithms. The proposed algorithms were simulated using NetBeans IDE 7.4 with JRE 1.7 running on Mac OS 10.9 on an Apple iMac. The CPU used is an Intel Core i5 3470S quad-core processor at 2.9 GHz, with 8GB of DDR3 RAM; the GPU used is an NVIDIA GeForce GTX 660M.

4.1 Example 1 – standard BIDDFS vs. optimised BIDDFS

First, the performance of the optimised BIDDFS was simulated for different parameters, e.g., field size and obstacles. The first set of simulations measures the performance gains of the optimised BIDDFS on open maps, i.e., no obstacles are presented on the map, allowing all nodes within the expansion area to be expanded, as the presence of obstacles will not allow that node to be expanded. This simulation is performed on maps of sizes between 25 × 25 nodes and 75 × 75 nodes. The threshold is measured to gauge the number of nodes expanded – the higher the threshold, the larger the expansion area, the more nodes are expanded. Table 1 tabulates the results of these simulations.

Comparing the algorithms on the 25 × 25 map, the optimised implementation is more than 2,336.33 times faster than the standard implementation. This is credited to the significantly reduced redundancy in cost calculation, including the removal of separate variables for OPEN and CLOSED sets. The threshold number is also much lower on the fast implementation, since it scans and identifies the boundary nodes and hence it is able to streamline expansion of boundary nodes. Scanning boundary nodes enables the algorithm to identify all possible boundary nodes for node expansion, eliminating any possibility of node re-expansion.

Table 1 Results comparing optimised BIDDFS to standard BIDDFS on open maps

Size	*Type*	*Time (s)*	*Threshold*
25 × 25	Standard	28.828	617
	Optimised	0.012339	45
50 × 50	Standard	496.631	2,492
	Optimised	0.041683	95
75 × 75	Standard	2798.105	5,617
	Optimised	0.088793	145

On the 75 × 75 map, optimised BIDDFS is able to register 31,512.67 times improved pathfinding time compared to the standard implementation on a map nine times larger than the 25 × 25 map. Node expansion is the greatest contributor to pathfinding time and memory efficiencies. This means that an improvement on a single number of node expansion can greatly alter the whole efficiency of pathfinding. The optimisations made by reducing the number of variables and calculations performed on node affects the whole map.

4.2 Example 2 – fast bidirectional BIDDFS vs. parallel bidirectional BIDDFS

This example measures and compares the time efficiency of the parallel bidirectional BIDDFS with the fast bidirectional BIDDFS proposed in Section 4. The maps used for this example is the same as the open maps used in example 1, with 25 × 25 and 75 × 75 nodes.

From the results in Table 2, a parallel bidirectional algorithm is able to register 551.25 times time improvement over the optimised BIDDFS on the 25 × 25 map. An increase in time efficiency of 9,041.95 times was recorded on the 75 × 75 map. Parallel bidirectional search algorithms searches from both the starting node and the goal node simultaneously, which is the main contributor to the increase in time efficiency recorded.

Table 2 Results comparing parallel bidirectional BIDDFS to the optimised BIDDFS on open

Size	*Algorithm*	*Time (s)*
25 × 25	Parallel bidirectional BIDDFS (PBD-BIDDFS)	6.185
	Fast bidirectional BIDDFS	0.01122
75 × 75	Parallel bidirectional BIDDFS (PBD-BIDDFS)	542.246
	Fast bidirectional BIDDFS	0.05997

4.3 Example 3 – comparison

The fast bidirectional BIDDFS algorithm is compared against other pathfinding algorithms, namely the IDDFS, Dijkstra's algorithm, A* search, IDA*, the standard bidirectional BIDDFS (BD-BIDDFS), and the fast bidirectional BIDDFS (fast BD-BIDDFS). Simulations are performed on open maps of increasing sizes from 10 × 10 to 30 × 30, with the starting and goal nodes placed at opposite ends of the map, giving an edge-to-edge search. The results are tabulated in Table 3. Bidirectional algorithms are denoted with the prefix 'BD-'. Furthermore, the results of the parallel new bidirectional A* (PNBA*) algorithm in Figure 2 in Rios and Chaimowicz (2011) simulated on the uniform cost grid pathfinding domain is compared and tabulated in Table 4.

Table 3 Results comparing fast bidirectional BIDDFS with other algorithms on open maps

Algorithm	*Time (s)*				
	10 × 10	*15 × 15*	*20 × 20*	*25 × 25*	*30 × 30*
Dijkstra's	0.247	0.487	0.604	0.930	1.229
A*	0.142	0.267	0.573	0.777	1.229
IDDFS	0.794	3.997	11.89	32.204	64.801
IDA*	0.408	3.083	8.210	27.051	63.797
BD-BIDDFS	0.222	0.691	2.347	6.199	13.072
Fast BD-BIDDFS	0.0055	0.0062	0.0088	0.0122	0.0161

Table 4 Results comparing the fast bidirectional BIDDFS with the PNBA* algorithm

Size	*Algorithm*	*Time (s)*
40 × 40	PNBA* (Rios and Chaimowicz, 2011)	0.1
	Fast bidirectional BIDDFS	0.0202
50 × 50	PNBA*	0.2
	Fast bidirectional BIDDFS	0.0286
60 × 60	PNBA*	0.5
	Fast bidirectional BIDDFS	0.0373

Since an edge-to-edge search is used, the majority of the map (> 80%) has to be expanded to reach the goal. With informed pathfinding algorithms having a slower single node expansion speed due to its heuristic function calculation, it is possible for uninformed searches to be as fast as an informed search (see Dijkstra's algorithm vs. A* search at 30 × 30 map). This means that an informed search is possible to be slower than an uninformed search when the same number of nodes is expanded during pathfinding.

The fast bidirectional BIDDFS up to 4,024 times faster than the IDDFS on the 30 × 30 map. This is credited to two factors in addition to bidirectional searching. Firstly, the fast bidirectional BIDDFS is based off the optimised BIDDFS, where its reduced read and write cycles on less variables yielded savings in time taken. Secondly, while standard bidirectional algorithms expands nodes originating from the starting node and goal node back and forth, parallel bidirectional searching expands nodes from the starting node and goal node simultaneously, increasing pathfinding time efficiency.

The fast bidirectional BIDDFS was also faster than the PNBA* in all simulations. The PNBA* registers a steeper increase in pathfinding time with map size. The parallel bidirectional BIDDFS was up to 13.4 times faster than the PNBA* when simulated on a 60 by 60 node map. The parallel bidirectional BIDDFS was faster than the PNBA* is due to the algorithm being an optimised algorithm for simplified searching, where the optimised BIDDFS had drastic increase in pathfinding speeds.

4.4 Example 4 – image-converted map performance measures

This section presents simulations performed on two image maps – a 50 × 50 node maze (see Figure 6) and a 250 × 250 node hotel floor plan (see Figure 7). This aims to allow the algorithm to simulate on real-world maps. A fast Dijkstra's algorithm is programmed for comparison with the optimised BIDDFS. For both maps, the starting node is placed at the upper left side of the map and the goal node at the lower right side of the map.

Figure 6 The 50 × 50 maze

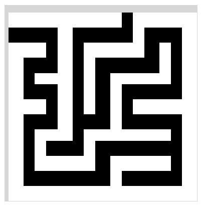


Figure 7 The 250 × 250 hotel floor plan

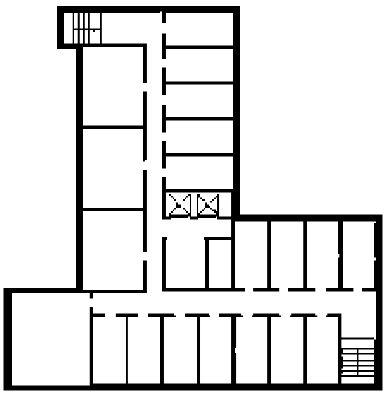


Table 5 Results comparing optimised BIDDFS to standard BIDDFS on image-converted maps

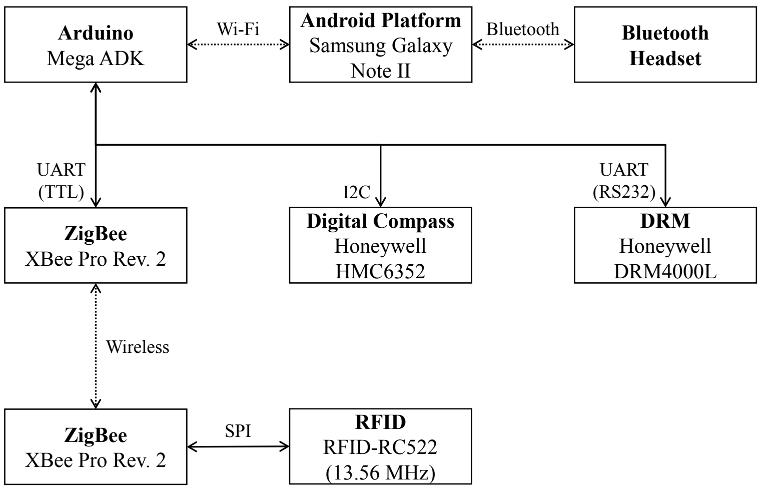
Map	*Algorithm*	*Time (s)*
Maze	Dijkstra's	0.126627
	Optimised BIDDFS	0.115573
	Fast BD-BIDDFS	0.019341
Hotel	Dijkstra's	2.041175
	Optimised BIDDFS	1.856176
	Fast BD-BIDDFS	0.843959

Results of this example are tabulated in Table 5. On the maze map, the optimised BIDDFS was 10% faster Dijkstra's algorithm. However, the fast bidirectional BIDDFS was 5.97 times faster than the optimised BIDDFS. On the hotel floor plan map, the optimised BIDDFS was 9% faster than the Dijkstra's algorithm. The fast bidirectional BIDDFS was 2.20 times faster than the optimised BIDDFS on the same map. The optimised BIDDFS was able to record a faster node expansion because the BIDDFS have fewer memory read and write cycles than the Dijkstra's algorithm.

5 Navigation for visually impaired people

While human navigation is often associated with the sense of sight to recognise directions, the visually impaired people are devoid of their sense of sight. Hence, the visually impaired often use navigational aids such as the white cane to support their navigation. A navigation system for the visually impaired can be built by converting the visual interface of a typical navigation with audio or tactile feedbacks. Therefore, a navigation system using Android device for the visually impaired can be built by referring to the block diagram in Figure 8.

Figure 8 Example of the system block diagram for the proposed navigation system



Note: Wireless interfaces in dotted arrows.

Positioning of the navigation system is provided via the radio frequency identification (RFID) (NXP Semiconductors, 2014) transponder. A Mifare RFID-RC522 is recommended for this system. RFID transponders are places on the navigation map as waypoints or point of interests (POIs), ensuring that the user is treading on the correct path. Therefore, increasing the number of RFID transponders in an area will increase positioning accuracy. With an array of RFID transponders placed on a map, a wireless many-to-one connection is required for the transponders to communicate with the user's device. Therefore, ZigBee (Digi International, 2008) is used for the wireless interfacing of RFID transponders. The XBee Pro Series 2 is used for the system and they communicate through transistor-transistor logic (TTL) serial.

While RFID provides absolute positioning (based on map), relative positioning (based on user's location) can be provided by a dead reckoning module (DRM) (Honeywell, 2011). The Honeywell DRM4000L can be used. To provide orientation data for relative positioning, a digital compass such as the Honeywell HMC6352 (Honeywell, 2006) can be implemented. This means that the RFID transponder determines the user's initial location for pathfinding to start, and that the DRM and digital compass tracks the movement of the user during navigation. At the same time, the user's position can be enhanced by allowing the user to tread on the RFID transponders until the destination is reached. The user interfaces the system using voice input to the system to specify the destination for pathfinding. Once the route has been found, guidance is provided in a visually impaired navigation system through audio feedbacks (e.g., text-to-speech). Therefore, a processing device is required in a navigation system to receive positioning data, perform pathfinding, and guide the user. An example of a processing device is an Android smartphone, in this case, a Samsung Galaxy Note II (Samsung Electronics Co. Ltd., 2014).

An Arduino is used to collect the positioning data sent by positioning devices (RFID transponder, DRM, and digital compass) to be sent to the Android device, because the positioning devices interfaces through different serial methods. To interface with these devices, an Arduino (2014) Mega ADK is recommended due to its multiple serial ports supporting multiple inputs and outputs. To communicate wirelessly to the Android device, a Wi-Fi shield is attached to the Arduino.

To use an Android smartphone as a processing device, a navigation application is programmed and installed onto the device. This application contains pathfinding and the interfacing source codes required to communicate with the system. An implemented pathfinding algorithm is imperative to ensure that the navigation system is able to provide the best route for the visually impaired user's navigation. Positioning devices connected to the Arduino interfaces to the Android device through user datagram protocol (UDP) over Wi-Fi, and audio feedback can be channelled through a Bluetooth headset. To allow communication with the visually impaired, the programmer to take advantage of the Google's text-to-speech application programming interface (API) (Google, 2014b) for speech synthesis for the user's guidance, and Google's speech-to-text API (Google, 2014a) for speech recognition of the user's input commands.

The pathfinding algorithms proposed in this paper, especially the fast bidirectional BIDDFS, are suitable for the application of this navigation system proposal. The proposed algorithms are to be programmed onto the Android platform of the system in Figure 8. Being completely programmed in the Android-native Java means that the algorithm is able to perform natively for that platform without the need for external compilers, guaranteeing the results demonstrated in this paper when it is implemented. It also means that the program in NetBeans can be directly ported onto this platform. The programmed algorithm will use data from the connected positioning devices to ascertain the user's location and orientation before pathfinding commences. Once pathfinding completes, the result will be used for the user's guidance through synthesised speech. Using an uninformed pathfinding algorithm allows the system to work in the absence of a map, therefore, the visually impaired user can freely navigate even in unknown environments.

6 Conclusions

This paper presented a navigation system for the visually impaired. The navigation system uses an Android device for processing. The pathfinding algorithms proposed for the Android device were programmed in Java to ensure that the algorithms ran natively. Firstly, the BIDDFS was extended for bidirectional searching in this paper, proposing the bidirectional BIDDFS. The bidirectional BIDDFS was faster than the IDDFS and the IDA* during experiments due to expansions from the starting and goal node. Fast searching was investigated for the BIDDFS, the BIDDFS was optimised for fast searching and the optimised BIDDFS was proposed. The optimised BIDDFS performed significantly faster than the standard BIDDFS with the reduction of read and write cycles. Fast searching was applied on the bidirectional BIDDFS using Java's parallel structure and thus proposing the fast bidirectional BIDDFS. The fast bidirectional BIDDFS was compared against the parallel bidirectional BIDDFS, and the fast approach made the fast bidirectional BIDDFS much faster than the parallel bidirectional BIDDFS.

Acknowledgements

This project has been supported by the Malaysian Ministry of Science, Technology and Innovation (MOSTI) eScience Fund number 01-02-16-SF0024.

References

Arduino (2014) *Arduino – ArduinoBoardMegaADK* [online] http://arduino.cc/en/Main/ArduinoBoardMegaADK (accessed 14 April 2014).

Björnsson, Y., Enzenberger, M., Holte, R.C. and Schaeffer, J. (2005) 'Fringe search: beating A* at pathfinding on game maps', Paper presented at the *Proceedings of IEEE Symposium on Computational Intelligence and Games*.

Bulitko, V., Björnsson, Y., Sturtevant, N. and Lawrence, R. (2011) 'Real-time heuristic search for pathfinding in video games', in González-Calero, P.A. and Gómez-Martín, M.A. (Eds.): *Artificial Intelligence for Computer Games*, pp.1–30, Springer, New York.

Cai, C., Yan, L. and Tie-Song, L. (2011) 'KM-A* pathfinding algorithm based on hierarchical clustering and strengthened DB index criteria', Paper presented at the *2011 International Conference on Machine Learning and Cybernetics (ICMLC)*, 10–13 July.

Di Giampaolo, E. (2010) 'A passive-RFID based indoor navigation system for visually impaired people', Paper presented at the *2010 3rd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL)*, 7–10 November.

Digi International (2008) *XBee™ ZNet 2.5/XBee-PRO™ ZNet 2.5 OEM RF Modules Datasheet*, Datasheet, 2/11/2008.

Dijkstra, E.W. (1959) 'A note on two problems in connexion with graphs', *Numerische Mathematik*, Vol. 1, No. 1, pp.269–271.

Folmer, E., Yuan, B., Carr, D. and Sapre, M. (2009) 'TextSL: a command-based virtual world interface for the visually impaired', Paper presented at the *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility*, Pittsburgh, Pennsylvania, USA.

Google (2014a) *android.speech | Android Developers* [online] http://developer.android.com/reference/android/speech/package-summary.html (accessed 18 April 2014).

Google (2014b) *TextToSpeech | Android Developers* [online] http://developer.android.com/reference/android/speech/tts/TextToSpeech.html (accessed 18 April 2014).

Honeywell (2006) *Digital Compass Solution – HMC6352 Datasheet*, January 2006.

Honeywell (2011) *DRM™4000L Dead Reckoning Module Datasheet*, September 2014.

Hua, W., Marshall, A. and Wai, Y. (2007) 'Path planning and following algorithms in an indoor navigation model for visually impaired', Paper presented at the *Second International Conference on Internet Monitoring and Protection, 2007, ICIMP 2007*, 1–5 July.

Khantanpoka, K. and Chinnasarn, K. (2009) 'Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer', Paper presented at the *Eighth International Symposium on Natural Language Processing*, Bangkok, Thailand.

Korf, R.E. (1985) 'Depth-first iterative-deepening: an optimal admissible tree search', *Artif. Intell.*, Vol. 27, No. 1, pp.97–109, DOI: 10.1016/0004-3702(85)90084-0.

Leigh, R., Louis, S.J. and Miles, C. (2007) 'Using a genetic algorithm to explore A*-like pathfinding algorithms', Paper presented at the *IEEE Symposium on Computational Intelligence and Games, 2007, CIG 2007*, 1–5 April.

Lim, K.L., Seng, K.P., Yeong, L.S., Ch'ng, S.I. and Ang, L-M. (2013) 'The boundary iterative-deepening depth-first search algorithm', Paper presented at the *Second International Conference on Advances in Computer and Information Technology – ACIT 2013*, Kuala Lumpur, Malaysia.

NXP Semiconductors (2014) *MFRC522 – Standard 3V MIFARE Reader Solution Product data Sheet*, Product Data Sheet, 17 September 2014.

Oktem, R., Aydin, E. and Cagiltay, N.E. (2008) 'An indoor navigation aid designed for visually impaired people', Paper presented at the *34th Annual Conference of IEEE Industrial Electronics, 2008, IECON 2008*, 10–13 November.

Oracle Corporation (2014) *Learn about Java Technology* [online] http://www.java.com/en/about/ (accessed 18 April 2014).

Rios, L.H.O. and Chaimowicz, L. (2011) 'PNBA*: a parallel bidirectional heuristic search algorithm', Paper presented at the *Proceedings of the XXXI Congresso da Sociedade Brasileira de Computação*, Natal, Brazil.

Samsung Electronics Co. Ltd. (2014) *Samsung GALAXY Note2 – Samsung Mobile* [online] http://www.samsung.com/global/microsite/galaxynote/note2/spec.html (accessed 15 April 2014).

Sturtevant, N.R. (2012) 'Benchmarks for grid-based pathfinding', *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2, pp.144–148.

Wu, X. and Zhang, S. (2011) 'The study and application of artificial intelligence pathfinding algorithm in game domain', Paper presented at the *International Conference on Computer Science and Service Systems*, Nanjing, China.