International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

A Novel Quad Harmony Search Algorithm for Grid-based Path Finding

Regular Paper

Saso Koceski[1,*], Stojanche Panov[1], Natasa Koceska[1], Pierluigi Beomonte Zobel[2] and Francesco Durante[2]

1 Faculty of Computer Science, University Goce Delcev, Stip, Macedonia
2 Department of Industrial and Information Engineering and Economy, DIIIE, University of L'Aquila, Italy
* Corresponding author E-mail: saso.koceski@ugd.edu.mk

Received 27 Oct 2013; Accepted 14 Jul 2014

DOI: 10.5772/58875

Abstract A novel approach to the problem of grid-based path finding has been introduced. The method is a block-based search algorithm, founded on the bases of two algorithms, namely the quad-tree algorithm, which offered a great opportunity for decreasing the time needed to compute the solution, and the harmony search (HS) algorithm, a meta-heuristic algorithm used to obtain the optimal solution. This quad HS algorithm uses the quad-tree decomposition of free space in the grid to mark the free areas and treat them as a single node, which greatly improves the execution. The results of the quad HS algorithm have been compared to other meta-heuristic algorithms, i.e., ant colony, genetic algorithm, particle swarm optimization and simulated annealing, and it was proved to obtain the best results in terms of time and giving the optimal path.

Keywords Heuristic Algorithms, Artificial Intelligence, Computational Intelligence, Optimization, Path Planning

1. Introduction

Throughout the past few decades, the global interest of researchers has been in the grid-based graph representations and path-planning problems, since they have been shown to be of great significance for many practical applications and research studies. Several of them were utilized in response to problems in the areas of computer vision, medical informatics, CAD/CAM-design, gaming and robotics. All these problems and applications become more challenging if they need to be solved in real-time. The algorithms employed for grid-based path finding can fall into two categories: deterministic and meta-heuristic. Because deterministic algorithms have many computational drawbacks, such as gradient information, great dependency on the initial value and often huge memory requirements, meta-heuristic algorithms have been considered as a feasible alternative. Meta-heuristic algorithms can be formulated as an iterative process, utilizing rules and randomness to improve the candidate solutions in order to efficiently find a near-optimal solution. Some of the meta-heuristic algorithms employed to solve the path-finding problems are: tabu search (TS), artificial neural network (ANN), genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO) and simulated annealing (SA). Although it has not always been possible to find an optimal path in grid-based environments using such algorithms, these researches have served as a great

example and a catalyst for finding alternatives to existing algorithms, thus providing either better time or memory complexities - key points of exploration to be seriously considered. Such research proved that these algorithms obtain poor results and a lack optimality when they have either very high or very low occupancy with obstacles in the working environment [1].

A new meta-heuristic algorithm called harmony search (HS), intended to overcome optimization and search problems, has been proposed by Geem et al. [2]. This algorithm utilizes the musician's experience in jazz improvisation to obtain an optimal solution through an iterative process. When HS has been evaluated against other meta-heuristic algorithms, a conclusion has been drawn that the HS algorithm performs better in terms of diversification and intensification concepts [3].

This means that HS has a randomization phase that is more efficient in exploring the search space, whilst not moving too far from possible good solutions and avoiding convergence with local optimal solutions. It is also noted that the greater the probability of the harmony memory consideration rate (HMCR) parameter, the faster the convergence to the best solution will be. The implementation of the HS is a lot easier and simpler than that of the other meta-heuristic algorithms, and the sensitivity to the chosen solutions will not be affected by fine tuning the parameters of the algorithm to obtain good solutions. It is also possible to parallelize the algorithm, which can provide better implementation with better efficiency [3].

This algorithm has been widely used in many engineering problems, such as internet routing [4], robotics [5] and web page clustering [6], but also in other real-world applications [7], electrical engineering [8], civil engineering [9], mechanical engineering [10], bioinformatics [11] and medical applications [12]. It can be easily adapted to any optimization matters according to one's needs. There have been several uses of this algorithm in motion planning and visual tracking [13], but to the best of our knowledge none of the previous researches have applied this algorithm to the grid-based graph search problem.

In mobile robotics, the path-finding problem in a grid-based environment could be defined as follows: where there is a set of vertices V, a set of edges E between these vertices, a start state and a goal state, an optimal route ought to be established between the start state and the goal state in a finite period of time – the planning time.

The edges between the vertices in the environment have a concrete cost assigned to them. A solution cost is the cost obtained by traversing the vertices of the computed path from the initial point to the destination. The algorithm that will provide this functionality ought to give a feasible or optimal path between two points in a finite time, supposing this path exists. To perform in such way, the hypothesis of safe exploration of the search problems needs to be established. Concretely, edges have finite costs; furthermore, where there are states s1, s2 and s3, if s1 and s2 are connected and s2 and s3 are connected, then s1 and s3 are connected.

There have been many attempts to obtain acceptable solutions which perform equally well with reduced time. Formally, all meta-heuristic algorithms mentioned in this chapter are applicable to path-finding problems in grid-based environments. However, they present various drawbacks depending on the environment structure. Many of them tend to become stuck in a local optimum of some sort, so that the result given is not optimal. This is especially evident in grids with the decreasing of the free space for robot movement. This fact makes them impractical for real-time applications as well as for remote control.

The main aim of this work is to propose a new algorithm capable of finding the optimal path (if one exists) in a grid-based environment at reduced time, compared to other meta-heuristic algorithms. This novel algorithm will prove its adaptability towards various types of real-world environments, having various percentages of obstacles and different sizes, and to prove its speed of execution and accuracy as well.

The proposed solution first separates the search space into smaller rectangles using the quad-tree (QT) algorithm, and then calls itself on these sub-problems recursively. During the search space division, rectangles in the search space are marked (coloured) and the fields with the same number (colour) are furthermore treated as a single node. This helps to create a reduced graph together with an adjacency list of the neighbouring rectangles, on which the HS approach is applied in order to find the shortest path between two given cells in the grid. The performances of the proposed algorithm have been evaluated empirically and using simulations. The obtained results confirm that the proposed quad harmony search (QHS) algorithm is superior compared to known meta-heuristic algorithms applied in the same environment and conditions.

2. Related work

Many meta-heuristic algorithms have been developed and applied for solving NP-complete problems such as the path-finding problem, mainly because they are robust enough and capable of producing acceptable solutions very quickly.

Particle swarm optimization (PSO) [14] is a population-based meta-heuristic algorithm inspired by the swarm behaviour of fish and bird schooling in the natural world. The algorithm explores the space of a fitness function by regulating the paths of individual searching agents, commonly known as particles. Besides the original algorithm, many PSO variants have been recently developed and applied to the grid-based robot path planning. For example, in [15], the paper proposes an approach using PSO in a shortest path-finding problem, which gave better results than algorithms that used the genetic algorithm (GA) as an alternative to obtain the optimal route. This approach also has the advantage of removing loops in providing the best possible path. The path-planning algorithm presented in [16], where PSO is applied in the path-planning problem for a mobile robot, the MAKLINK graph is firstly constructed to describe the search space of the robot, then the Dijkstra algorithm is applied to provide the shortest distance from the start node to the end node, and finally the PSO is used to give the optimal path. Results have shown that this approach is applicable in real-time mobile robot navigation. The same authors later presented a modified version of the approach in the previously published paper [17]. In order to escape the plunging into the local minimum, they added a mutation operator, which led to greater speed in performance in the early phase of the algorithm. The hybrid algorithm described in [18], known as CIPSO, includes a combination of several techniques, including artificial immune system (AIS), chaos operator and the PSO algorithm. This approach has been shown to give better results than the GA and the PSO in terms of optimality of route and the time execution of the algorithm.

Simulated annealing (SA) [19] is a meta-heuristic algorithm that is often utilized in problems that require the obtaining of a solution in a certain amount of time, rather than requiring the optimal solution. The slow cooling that appears in the process of annealing (a process known in metallurgy) is implemented in the algorithm, as it performs a slow decrease, respecting the probability of accepting worse solutions in the search space. It has also found its use in several mobile robot path-planning problems, and it has also been combined with other techniques. In [20], the SA algorithm is applied to the robot path-planning problem to boost the artificial potential field (APF) approach and avoid plunging into local minima, a method frequently used in real-time problems. This algorithm has been shown to be greatly effective in local and global path finding. The paper [21] describes the same technique and applies this method to soccer robots' path planning, and has also proven the validity of this approach. Also, a simulated annealing neural network has been introduced in [22], which is used to describe the obstacles in the robot's environment. This path planner has been effectively applied to several kinds of robots, like flying robots and snake robots.

The ant colony (AC) algorithm's probabilistic nature gives well-approximated solutions and is often used as an optimization technique [23]. It is mainly based on the behaviour of ants as they search for food and find a path from their colony. After finding food, they return to their colony, leaving pheromone trails. These pheromone trails tend to evaporate over a given period, which is due to the concrete time an ant has to travel to the food. Hence, the shorter the path, the less evaporation will happen and the pheromone density will increase. This phenomenon is the reason why this algorithm cannot get stuck in a locally optimal solution. When one ant finds food by following a given (shorter) path, the other ants will be more likely to follow the same path. The paper [24] shows the application of the AC to the two-dimensional robot path-planning problem, which solves the problem of the local optima and increases search speed. In the method proposed in [25], called SACOdm, the decision-making process is improved by storing the existing distances in memory, since the ant colony algorithm itself cannot remember the visited nodes, which produces a speed-up of around 10 in several cases [25]. Then, the optimal path is selected using fuzzy interference systems, applying the simple tuning algorithm. An improved augmented ant colony algorithm has been introduced in [26], which decreases the time for the execution of the initialization phase by adding the heuristic probability function in this phase, which solves the precocity of the algorithm. Results have shown that this technique has better performance and greater optimization capabilities than the traditional augmented ant colony algorithm. The AC algorithm has also been applied to the robot path-planning problem in a dynamic environment with dynamically appearing obstacles [27]. This implementation uses two different re-initialization schemes.

Another meta-heuristic algorithm which is also used for search and optimization is the genetic algorithm [28], which is a search heuristic mirroring process of natural evolution, such as the processes of crossover, mutation, selection and inheritance. Here, the evolution is headed towards better solutions, typically encoded with 0s and 1s, but there can be other types of encodings. This algorithm finishes when a certain number of iterations is reached, or a given value of the fitness function is obtained; hence, the solution to the problem is the finally generated generation. A knowledge-based genetic algorithm (GA) applied to the mobile robot path-planning problem has been introduced in [29], applying its domain knowledge to its operators. This effective technique has demonstrated its utility in obtaining the optimal or near-optimal path of a mobile robot both in

static and dynamic environments. The global path planning using neural network and genetic algorithm presented in [30] uses the neural network to construct the search space (environment) of the robot in order to establish a connection between a collision avoidance path and the output of the neural network, and then this information, along with the information for obtaining the optimal route, is fed to the fitness function of the genetic algorithm. Also, an effective method to obtain the optimal route of a mobile robot is the chaos genetic algorithm [31], so that when the chaos operation is added to the genetic algorithm, it decreases the time needed for convergence and performs well in avoiding collisions with obstacles. The paper [32] has described the implementation of the GA on the path-planning problem by using four neighbour movements in a grid-based environment.

3. The harmony search algorithm

Harmony search (HS) is a music-inspired technique which mimics the process of improvisation of jazz musicians; it is analogous to the musicians' examination of all the possible combinations of musical pitches they have remembered and their recalling them from memory. In such a fashion, this algorithm is applied to exploring such combinations with the aim of solving various optimization problems. It is a probabilistic technique which finds the optimal solution in several different stages.

1. First, the harmony memory (HM) is initialized. This process is performed by generating random numbers in a specified range, and this memory would look as follows:

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^N & x_2^N & \cdots & x_n^N \end{bmatrix} \quad (1)$$

where n is the dimension of the candidate vectors and N is the total memory size.

2. Secondly, a new candidate solution is improvised, namely $\begin{bmatrix} x_1' & x_2' & \dots & x_n' \end{bmatrix}$, based on the harmony memory consideration rate (HMCR), which is defined as the probability of the new candidate solution being chosen from the HM. So, in a similar way, the probability of the candidate solution to be selected randomly is (1-HMCR). If a member of the candidate solution is chosen from the harmony memory, it can be changed further due to the pitch adjustment rate (PAR), which is the probability of the candidate being mutated.

3. Thirdly, the new candidate solution is compared to the other candidate solutions stored in the HM. If the new candidate solution provides a better result than the worst candidate solution whilst evaluating the function, it is stored in the memory and the worst solution is discarded from the memory. Otherwise, this new candidate solution is not considered in future, and the previous processes are repeated. The algorithm stops when a certain termination criterion is reached, such as number of iterations.

4. Quad harmony search (QHS) algorithm description

In this paper, we introduce a novel approach, based on the utilization of the HS algorithm, to the problem of graph search in grid-based environments [1, 33]. The grids that have been examined are of a rectangular shape, having square-sized cells of fixed height and width, and the agent was able to move in four directions, marked as north (N), east (E), south (S) and west (W). For the purposes of algorithm optimization, we use a search space reducing technique in order to speed up the process of finding a solution, and provide a better convergence in fewer iterations and with smaller memory size. This technique has been widely used in geometry-related problems, but for the main goals of this paper, it proved to raise an important issue in terms of time and space complexity. It is commonly known as a quad-tree algorithm.

The quad-tree (QT) algorithm is based on a dividing technique which separates the search space into four blocks, i.e., sub-problems, and then calls itself on these sub-problems recursively. The algorithm stops with the recursive calls when it meets certain defined criteria. For the main goals of this research, the QT technique is used in a way which marks (colours) certain squares in the search space, and the fields which bear the same number (colour) are furthermore treated as a single node. The technique is adapted to this research as follows:

1. Take as input the full grid to be examined.
2. If the size of the grid is 1x1, if it is empty, mark with number (colour), else mark with -1.
3. Divide the search space into four square-like subspaces.
4. Check each of the square-like subspaces for existence of obstacles.
   a. If none of the subspaces have obstacles, mark them all as a single number (colour).
   b. If two adjacent square-like subspaces do not have obstacles, mark them with the same colour. For the other two, repeat recursively, starting from step 2.
   c. If the previous two are not satisfied, check if any subspace contains obstacles. The ones that do not contain obstacles mark them with the same number. For the ones that contain obstacles, repeat recursively from step 2.

The running of this algorithm on an example grid and its result are shown in Figure 1 and Figure 2.

After the grid is labelled with the proper numbers, a reduced graph is constructed using the previous separation, and an adjacency list is obtained using the neighbouring rectangles (squares). This approach is extremely efficient when it comes to searching through a maze that has a large amount of free areas and the percentage of obstacles in the grid is relatively small.

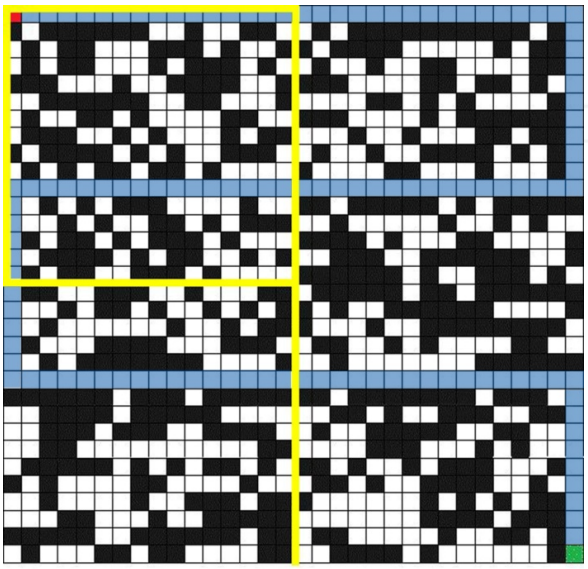

Figure 1. Example of applying QT on a grid. Red colour indicates start, whereas green colour indicates end. Yellow colour presents the partitioning of the algorithm for the quadrant of interest and the blue colour displays the optimal path for that given grid.

Once the adjacency list is constructed, the proper input data is ready to be fed to the HS algorithm.

1	1	3	3	6	6	7	7	19	19	20	20	24	24	26	26
-1	2	-1	-1	-1	-1	-1	-1	19	19	-1	21	25	-1	-1	27
4	-1	5	-1	-1	8	9	9	-1	-1	-1	23	-1	28	29	-1
4	-1	5	-1	-1	8	9	9	-1	22	-1	-1	-1	28	-1	30
-1	-1	-1	11	14	14	-1	15	31	-1	-1	-1	-1	36	38	-1
10	10	-1	-1	-1	-1	-1	-1	31	32	-1	-1	37	36	38	-1
12	-1	13	13	-1	-1	17	-1	33	33	35	35	-1	-1	40	40
12	-1	-1	-1	16	16	-1	18	-1	34	35	35	-1	39	-1	-1
-1	41	-1	-1	-1	-1	48	-1	65	-1	66	66	-1	-1	-1	72
42	-1	43	-1	47	-1	-1	49	-1	-1	67	-1	71	71	73	72
44	44	45	45	50	50	52	52	68	68	70	70	74	74	76	76
44	44	-1	46	-1	51	-1	-1	-1	69	70	70	75	-1	76	76
53	53	55	-1	59	-1	61	61	-1	-1	77	-1	82	-1	-1	-1
54	-1	55	-1	-1	60	-1	62	-1	-1	-1	78	-1	83	84	84
56	56	-1	-1	-1	-1	64	-1	79	-1	-1	80	85	-1	87	-1
57	-1	58	-1	-1	63	64	-1	-1	-1	81	-1	85	86	87	88

Figure 2. The marked grid of the yellow coloured upper-left quadrant in Figure 1

The memory consideration rate of the algorithm (HMCR), as defined in the previous section, is the probability for a candidate solution to be chosen from the harmony memory (HM). The greater the consideration rate, the greater the probability for a candidate solution to be selected from the harmony memory (HM), as in Eq. 2 [2]:

$$x_i^{'} \leftarrow \begin{cases} x_i^{'} \in \left[x_i^1 \quad x_i^2 \quad \cdots \quad x_i^{HMS} \right] & w.p. \quad HMCR \\ x_i^{'} \in X_i & w.p. \quad (1-HMCR) \end{cases} \tag{2}$$

where X_i is the set of possible values in range [1; *maximumNumberOfRectangles*], which means all possible values are dependent on the number of rectangles generated by quad-tree, denoted as *maximumNumberOfRectangles*. The equation states that, with a probability of HMCR, a candidate solution x_i' will be generated from the HM, and with a probability of (1-HMCR) it will be generated randomly from the set of possible values X_i.

Since the parameters are set correctly, the fitness function needs to be defined, which is used to obtain convergence to a desired and acceptable solution. In a single iteration of the evaluation, the following process is implemented (the pseudocode is available in Figure 3):

1. Declare a variable prevRect to track the previously visited rectangle.
2. Set prevRect to the start node in the candidate vector.
3. For each member in the vector:
   a. If we have reached the end node, increase the value of the fitness function by one and terminate this iteration.
   b. Declare variable now and set it to the value of prevRect.
   c. From all of the neighbours of the current rectangle, choose the next rectangle for exploration in the following manner: the next value in the candidate vector modulus number of the neighbours of the current rectangle. This number is the index of the next rectangle in the adjacency list.
   d. Increase the fitness function by one and if the end of the vector is not reached, return to step 3.a.

Thus, given the algorithm previously described, the formula for the fitness function ready to be minimized would be simply defined as:

$$f(.) = \sum_{d \in D} PathCost(d) \tag{3}$$

where d is a direction the algorithm is taking (a node in the maze), D is the set of directions (nodes) given by the random candidate solution (with the destination node as final element), and *PathCost(d)* is a predefined cost for taking the direction d and can be defined as follows:

```
int prevRect = starting_rectangle;
int fitnessValue = 0;
int const = 1;
foreach number in candidate_vector
begin
 if prevRect is ending_rectangle
 begin
        fitnessValue+=const;
        break;
 end
 int now = prevRect;
 int index =
        number%NumberOfNeighbours(now);
 int nextRect = Neighbours[index];
 prevRect = nextRect;
 fitnessValue+=const;
end
```

Figure 3. Pseudocode for the fitness function of the harmony search method

$$PathCost(d) = \begin{cases} 1, & d \text{ before reaching final node} \\ 0, & d \text{ after reaching final node} \end{cases} \quad (4)$$

When the end of the HS stage is reached, the path from the best resulting candidate vector previously obtained is constructed. This is done by retrieving the positions of the upper-left corners and dimensions of each of the rectangles present in the obtained solution, and by simple calculation of the minimum Manhattan distances between them, which greatly improves the performance of this algorithm.

5. Simulation

The developed QHS algorithm has been evaluated for global path planning in grid environments using simulations. Under the same conditions it was compared with other known meta-heuristic approaches: ant colony (ACO), genetic algorithm (GA), particle swarm optimization (PSO) and simulated annealing (SA). The simulation environment has been developed and implemented in Java programming language. All the evaluated algorithms were implemented as separate modules. An additional module was developed for generation of grid environments with a random filling percentage of obstacles. All the simulations were executed on a PC with an Intel® Core i5 processor with a frequency of 2.53 GHz, 4GB of RAM and 64-bit Windows 7 operating system.

In the simulation experiments, we used the following values for the QHS parameters: HMCR of 0.9, BW (bandwidth) of 0.2, PAR of 0.4, size of a candidate vector set to the number of labelled rectangles (squares), size of HM set to 10 and number of iterations (the termination criterion) set to 50.

The planning time, i.e., the time required to find the optimal path from a given start to a given end position was considered as a main evaluation parameter.

In the first simulation test, all the algorithms have been run on the example grid shown in Figure 1. For the given grid and the same start and goal position (marked with green and red respectively in Figure 1), the obtained results are shown in Table 1.

Algorithm	Planning Time (ms)
Quad Harmony Search	55
Ant Colony	5513
Genetic Algorithm	41525
Particle Swarm Optimization	40749
Simulated Annealing	37958

Table 1. Comparison of the planning times of the evaluated algorithms for the grid given in Figure 1

From the results, one can observe that the QHS algorithm has given the best results compared to the other algorithms whilst obtaining the optimal solution, while the genetic algorithm gave the worst results.

To test the dependency of the planning time on the grid size and the filling percentage of obstacles, several other types of grid environments, with sizes of 8 x 8, 16 x 16, 32 x 32, 64 x 64 and 128 x 128, were simulated and examined. They were clustered with different percentages of obstacles, starting at 10 % and finishing at 90 %, with a step of 10 %. For each size and percentage of obstacles, 10 different grid variants were simulated. For all grid variants $Vi\ (i=1..10)$ of the size $Sj,\ j \in \{8 \text{ x } 8,\ 16 \text{ x } 16,\ 32 \text{ x } 32,\ 64 \text{ x } 64,\ 128 \text{ x } 128\}$ and obstacle percentage $Pk,\ k \in \{10\ \%,\ 20\ \%,\ 30\ \%,\ 40\ \%,\ 50\ \%,\ 60\ \%,\ 70\ \%,\ 80\ \%,\ 90\ \%\}$ for the same start and goal positions, the planning times $PTi(Sj,Pk)$ were measured for all the algorithms.

The measured planning times for grids of a specific size and obstacle percentage were averaged and calculated for each of the algorithms as:

$$PT(Sj,Pk) = \frac{\sum_{i=1}^{N} PTi(Sj,Pk)}{N},\ N = 10 \quad (5)$$

The average planning times calculated using Equation 5, given in milliseconds, are presented on Table 2, Table 3, Table 4, Table 5 and Table 6 for each algorithm, respectively. The comparison of the tested algorithms for the generated grid variants is shown in Figures 4-8.

Obstacles (%)	Grid Sizes				
	8 x 8	16x16	32x32	64x64	128x128
10 %	12	23	51	367	5398
20 %	14	14	65	659	5101
30 %	15	23	83	930	9655
40 %	13	27	62	710	8732
50 %	14	19	57	108	962
60 %	12	22	38	70	472
70 %	11	23	19	51	389
80 %	9	8	13	28	172
90 %	8	15	14	21	115

Table 2. Planning time on different grid sizes and different percentage of obstacles using quad harmony search. Results are shown in milliseconds.

Obstacles (%)	Grid Sizes				
	8x8	16x16	32x32	64x64	128x128
10 %	795	2440	9501	16094	30903
20 %	1102	2861	8896	20145	36472
30 %	971	2007	7974	21538	33812
40 %	598	2811	5441	18668	29171
50 %	461	1702	5513	8640	22920
60 %	349	1131	3215	7859	21174
70 %	278	701	2701	7003	21854
80 %	334	818	2150	6634	20801
90 %	374	695	2585	6020	22563

Table 3. Tests on different grid sizes and different percentages of obstacles using ant colony. Results are shown in milliseconds.

Obstacles (%)	Grid Sizes				
	8x8	16x16	32x32	64x64	128x128
10 %	3171	9321	25540	1114136	4254514
20 %	3260	6290	38588	576558	3695286
30 %	3170	11200	33670	1607530	1942007
40 %	3140	9080	31529	1619849	1913206
50 %	2840	20040	41525	428831	1443848
60 %	2930	20996	36724	279093	561349
70 %	2990	13150	45172	267380	571141
80 %	2030	13852	54988	514670	634857
90 %	2820	15814	52294	268741	1616770

Table 4. Planning time on different grid sizes and different percentages of obstacles using genetic algorithm. Results are shown in milliseconds.

Obstacles (%)	Grid Sizes				
	8x8	16x16	32x32	64x64	128x128
10 %	2983	9757	24874	1073628	4325631
20 %	3142	6381	37736	589736	3716732
30 %	3267	12725	32846	1584763	1734847
40 %	2975	9120	34846	1473904	1489362
50 %	2591	10528	40749	424354	1538727
60 %	2381	21467	37635	256765	572334
70 %	2553	15332	44857	274556	572976
80 %	2147	14398	53874	509264	624558
90 %	2913	16385	52856	278346	998734

Table 5. Planning time on different grid sizes and different percentages of obstacles using particle swarm optimization. Results are shown in milliseconds

Obstacles (%)	Grid Sizes				
	8x8	16x16	32x32	64x64	128x128
10 %	3072	9843	25843	995469	4486763
20 %	3078	6428	38698	590174	3694576
30 %	3383	13745	30784	1477246	1487987
40 %	2856	10739	31874	1390523	1776724
50 %	2389	11483	37958	409583	1686820
60 %	2294	22857	34896	239974	588290
70 %	2677	14762	45898	298758	557927
80 %	2211	13879	55780	520887	634887
90 %	3141	15898	54872	308548	979375

Table 6. Planning time on different grid sizes and different percentage of obstacles using simulated annealing. Results are shown in milliseconds.

Figure 4. Comparison between planning times of quad harmony search, ant colony, genetic algorithm, particle swarm optimization and simulated annealing for 8 x 8 grid sizes

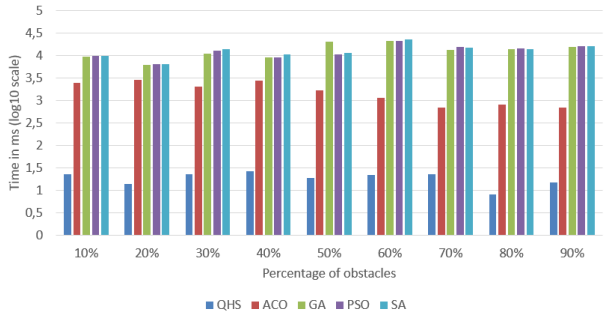

Figure 5. Comparison between planning times of quad harmony search, ant colony, genetic algorithm, particle swarm optimization and simulated annealing for 16 x 16 grid sizes

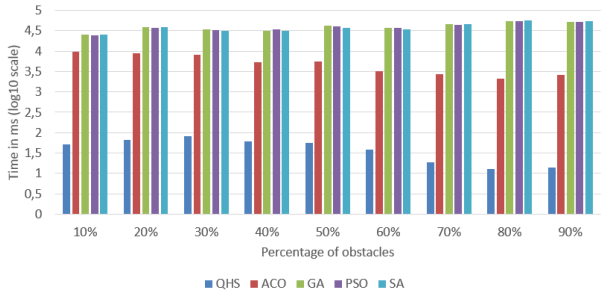

Figure 6. Comparison between planning times of quad harmony search, ant colony, genetic algorithm, particle swarm optimization and simulated annealing for 32 x 32 grid sizes

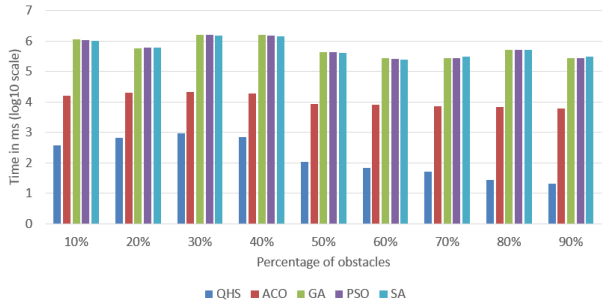

Figure 7. Comparison between planning times of quad harmony search, ant colony, genetic algorithm, particle swarm optimization and simulated annealing for 64 x 64 grid sizes

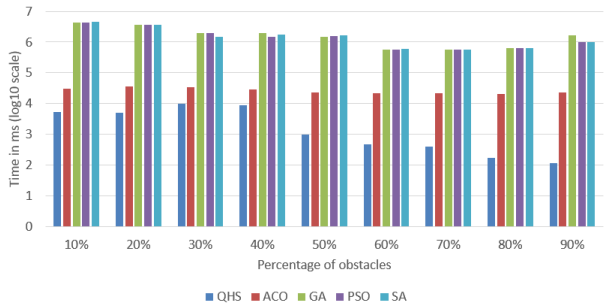

Figure 8. Comparison between planning times of quad harmony search, ant colony, genetic algorithm, particle swarm optimization and simulated annealing for 128 x 128 grid sizes

6. Experimental results

The developed algorithm has also been evaluated experimentally. The main aim of the experimental evaluation was to confirm the ability to generate an optimal and collision-free path of the developed algorithm. For experimental evaluation of the developed algorithm, a specific laboratory setup for indoor motion planning was used.

For this purpose, besides the path-planning phase, the system should also be able to perform robot localization and motion control. The system consists of three main components: mobile robot, control workstation and ceiling-mounted cameras for localization of the robot, obstacles and the target.

The three-wheel mobile robot used in our study is a modified version of the ARobot (Figure 9) [34]. The robot contains one Basic Stamp II controller from Paralax [35], and two coprocessors: PIC16F84 for motor control. The robot has the following sensors: sonar, two light sensors, temperature sensor, whisker sensors, PIR passive infrared motion detector, digital compass, R283-HOKUYO-LIDAR and sound output transducer. The robot has two 12-volt DC drive motors. These motors are regulated independently using PWM-controlled H-bridges. The robot also has optical encoders that enable determination of speed and position of the robot's wheels. All these components are placed in a lightweight aluminium construction with these dimensions; 10" x 10", 5" tall, with a payload capacity of 3 lbs.

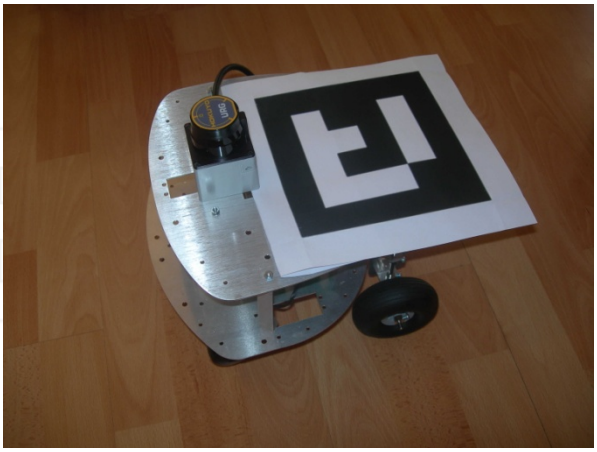

Figure 9. The robot used for experimental evaluation

The mobile robot is connected to the control workstation via wireless link using RF two-way radio modules. The workstation is running a program that is responsible for robot localization, path planning and motion control planning.

Considering the dimensions of the working environment and the necessity for real-time control, a multi-camera-based system for localization tasks has been employed. The working environment was virtually split into four overlapping regions and each of them has been fully covered by a ceiling-mounted web camera with a 90-degree field of view, sending 640 x 480 real-time videos to the command workstation for further processing. This way, all the obstacles (marked with white or green colour), the target (marked with blue) and the robot, marked with QR code, were captured with satisfactory resolution, and thus became easily detectable (Figure 10). The image processing task has been performed at the control workstation by the localization module, which takes video as input, detects the obstacles and generates the grid-based map. The obstacles were detected using an algorithm based on colour filtering and edge detection with a localization error of less than 2 cm.

Once created, the grid-based map is sent to the path-planning module, which employs the developed quad-tree harmony search algorithm to find the path from the start to the end point. At the end, in order to control the mobile robot, the control program running on the workstation sends direct commands to the robot actuators to perform the actual movement.

Figure 10. Real-world experimental environment (including obstacles and the robot)

In the experimental evaluation, the planning time is considered as the time required to find the optimal path from a given start to a given end position, after obtaining the grid-based map of the environment until the path calculation.

For the experiments, two grid sizes, 64 x 64 and 128 x 128, with 30 %, 40 % and 50 % obstacles were considered. Ten grid samples of each grid type were generated, and for each of them two different start and goal positions were analysed. In all the cases, QHS found the optimal path. The average planning times obtained experimentally are shown in Table 7.

Obstacles (%)	Grid Sizes			
	64x64		128x128	
	T	STD	T	STD
30 %	100	5	201	6
40 %	234	7	307	4
50 %	38	3	71	5

Table 7. Planning times and standard deviations (in milliseconds) obtained with the experimental evaluation of quad harmony search

All the results were verified for correctness using a deterministic algorithm, namely the breadth-first search algorithm. It was easily proved that all of the paths obtained by QHS have the exact same length with the paths provided by the breadth-first search algorithm.

7. Discussion

Analysing the simulation results provided by QHS algorithm, we can easily infer that at a lower percentage of obstacles (10 %-20 %) we mainly get faster planning times, because we use the QT algorithm to determine the free areas in the grid-based graph. This is also the case for a greater percentage of obstacles (70 %-90 %). However, in the range of 30 %-60 % of obstacles, we get longer planning times. This can also be explained in terms of the greater number of rectangles that have to be examined; therefore, we get greater memory size (HM), more possibilities to consider, and all these lead to a longer period spent exploring the optimal solution.

Comparing the results we have obtained with QHS to ant colony (ACO), genetic algorithm (GA), particle swarm optimization (PSO) and simulated annealing (SA), several crucial conclusions have been drawn. Ant colony proved to always give the optimal path, though it took longer to execute than QHS in 100 % of the test cases. Using the GA, PSO and SA, results show that the algorithm can always find a feasible solution, but on the examined test cases it never obtained the optimal solution. This means that GA, PSO and SA can easily be stuck in a local optima and thus not give the best possible solution. Also, in terms of time execution compared to QHS and ant colony, these three algorithms (GA, PSO and SA) gave poor results.

Thus, one can simply infer that the proposed QHS is the best fit to the problem of graph search in maze-like environments.

The results obtained with the experimental evaluation correspond to the simulated ones. Experimentally obtained planning times for optimal path calculation make the QHS suitable for real-time robot control and applicable to various real-world applications. Relatively small standard deviations prove the stable behaviour of the algorithm and its robustness.

8. Conclusion

In this research, we introduced a technique to find an optimal solution to a grid-based graph search using quad-tree decomposition to reduce the search space, and harmony search to find the optimal route between the connected areas, finishing by extracting the path using the Manhattan distance between the maze fields that belong to these areas. This algorithm was compared to ant colony, genetic algorithm, particle swarm optimization and simulated annealing and was proven to give the best results correlated to obtaining the optimal path. It has also been shown that our approach gave best results in mazes with lower percentages of obstacles, which means that it is suitable to use when one needs faster performance in these concrete cases. This gives an open opportunity for further researches and examinations of the application of this technique in various graph searches, and for the expansion of this approach to greater dimensions.

9. References

[1] Panov, S., & Koceski, S. (2013, June). Harmony search based algorithm for mobile robot global path planning. In 2nd Mediterranean Conference Embedded Computing (MECO), 2013 (pp. 168-171). IEEE.

[2] Yang, X. S. (2009). Harmony search as a metaheuristic algorithm. In Music-inspired Harmony Search Algorithm (pp. 1-14). Springer Berlin Heidelberg.

[3] Geem, Z. W. (2010). Recent Advances in Harmony Search Algorithm (Vol. 270). Springer Berlin Heidelberg.

[4] Forsati, R., Haghighat, A. T., & Mahdavi, M. (2008). Harmony search based algorithms for bandwidth-delay-constrained least-cost multicast routing. Computer Communications, 31(10), 2505-2519.

[5] Tangpattanakul, P., & Artrit, P. (2009, May). Minimum-time trajectory of robot manipulator using Harmony Search algorithm. In Sixth International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. (Vol. 1, pp. 354-357). IEEE.

[6] Forsati, R., Mahdavi, M., Kangavari, M., & Safarkhani, B. (2008, May). Web page clustering using harmony search optimization. In Canadian Conference on Electrical and Computer Engineering, 2008. CCECE 2008. (pp. 001601-001604). IEEE.

[7] Geem, Z. W. (2007). Optimal scheduling of multiple dam system using harmony search algorithm. In Computational and Ambient Intelligence (pp. 316-323). Springer Berlin Heidelberg.

[8] Kim, J. H., Geem, Z. W., & Kim, E. S. (2001). Parameter estimation of the nonlinear Muskingum model using harmony search. JAWRA Journal of the American Water Resources Association, 37(5), 1131-1138.

[9] Vasebi, A., Fesanghary, M., & Bathaee, S. M. T. (2007). Combined heat and power economic dispatch by harmony search algorithm. International Journal of Electrical Power & Energy Systems, 29(10), 713-719.

[10] Fesanghary, M., Mahdavi, M., Minary-Jolandan, M., & Alizadeh, Y. (2008). Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. Computer Methods in Applied Mechanics and Engineering, 197(33), 3080-3091.

[11] Mohsen, A. M., Khader, A. T., & Ramachandram, D. (2010). An optimization algorithm based on harmony search for RNA secondary structure prediction. In Recent Advances in Harmony Search Algorithm (pp. 163-174). Springer Berlin Heidelberg.

[12] Moh'd Alia, O., Mandava, R., & Aziz, M. E. (2011). A hybrid harmony search algorithm for MRI brain segmentation. Evolutionary Intelligence, 4(1), 31-49.

[13] Fourie, J., Mills, S., & Green, R. (2010). Harmony filter: a robust visual tracking system using the improved harmony search algorithm. Image and Vision Computing, 28(12), 1702-1716.

[14] Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimisation. In Proceedings of IEEE International Conference on Neural Networks, 1995.

[15] Mohemmed, A. W., Sahoo, N. C., & Geok, T. K. (2008). Solving shortest path problem using particle swarm optimization. Applied Soft Computing, 8(4), 1643-1653.

[16] Qin, Y. Q., Sun, D. B., Li, N., & Ma, Q. (2004). Path planning for mobile robot based on particle swarm optimization. Robot, 26(3), 222-225.

[17] Qin, Y. Q., Sun, D. B., Li, N., & Cen, Y. G. (2004, August). Path planning for mobile robot using the particle swarm optimization with mutation operator. In Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004. (Vol. 4, pp. 2473-2478). IEEE.

[18] Zhang, Y., Jun, Y., Wei, G., & Wu, L. (2010). Find multi-objective paths in stochastic networks via chaotic immune PSO. Expert Systems with Applications, 37(3), 1911-1919.

[19] Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing (pp. 7-15). Springer Netherlands.

[20] Zhu, Q., Yan, Y., & Xing, Z. (2006, October). Robot path planning based on artificial potential field approach with simulated annealing. In Sixth International Conference on Intelligent Systems Design and Applications, 2006. ISDA '06. (Vol. 2, pp. 622-627). IEEE.

[21] Zhang, P. Y., Lü, T. S., & Song, L. B. (2004). Soccer robot path planning based on the artificial potential field approach with simulated annealing. Robotica, 22(5), 563-566.

[22] Kroumov, V., Yu, J., & Shibayama, K. (2010). 3D path planning for mobile robots using simulated annealing neural network. International Journal of Innovative Computing, Information and Control, 6(7), 2885-2899.

[23] Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on (Vol. 2). IEEE.

[24] Liu, G., Li, T., Peng, Y., & Hou, X. (2005, July). The ant algorithm for solving robot path planning problem. In Third International Conference on Information Technology and Applications, 2005. ICITA 2005. (Vol. 2, pp. 25-27). IEEE.

[25] Garcia, M. A., Montiel, O., Castillo, O., Sepúlveda, R., & Melin, P. (2009). Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. Applied Soft Computing, 9(3), 1102-1110.

[26] Gao, M., Xu, J., & Tian, J. (2008, September). Mobile robot global path planning based on improved augment ant colony algorithm. In Second International Conference on Genetic and Evolutionary Computing, 2008. WGEC '08. (pp. 273-276). IEEE.

[27] Brand, M., Masuda, M., Wehner, N., & Yu, X. H. (2010, June). Ant colony optimization algorithm for robot path planning. In 2010 International Conference on Computer Design and Applications (ICCDA). (Vol. 3, pp. V3-436). IEEE.

[28] Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2), 65-85.

[29] Hu, Y., & Yang, S. X. (2004, April). A knowledge based genetic algorithm for path planning of a mobile robot. In Proceedings of 2004 IEEE International Conference on Robotics and Automation, 2004. ICRA '04. (Vol. 5, pp. 4350-4355). IEEE.

[30] Xin, D., Hua-hua, C., & Wei-kang, G. (2005). Neural network and genetic algorithm based global path planning in a static environment. Journal of Zhejiang University Science A, 6(6), 549-554.

[31] Gao, M., Xu, J., Tian, J., & Wu, H. (2008, October). Path planning for mobile robot based on chaos genetic algorithm. In Fourth International Conference on Natural Computation, 2008. ICNC '08. (Vol. 4, pp. 409-413). IEEE.

[32] Ismail, A. T., Sheta, A., & Al-Weshah, M. (2008). A mobile robot path planning using genetic algorithm in static environment. Journal of Computer Science, 4(4), 341.

[33] Panov, S., & Koceska, N. (2014). Global path planning in grid-based environments using novel meta-heuristic algorithm. In ICT Innovations 2013 (pp. 121-130). Springer International Publishing.

[34] http://www.arrickrobotics.com/arobot/ Accessed on October 2013.

[35] http://www.parallax.com Accessed on October 2013.