Best Path in Mountain Environment based on Parallel Hill Climbing Algorithm

Raja Masadeh[1]
Computer Science department
The World Islamic Sciences and Education University
Amman, Jordan

Ahmad Sharieh[2], Sanad Jamal[3]
Mais Haj Qasem[4]
Computer Science department
The University of Jordan
Amman, Jordan

Bayan Alsaaidah[5]
Computer Science department
Al-Balqa Applied University
Al-Salt, Jordan

***Abstract*—Heuristic search is a search process that uses domain knowledge in heuristic rules or procedures to direct the progress of a search algorithm. Hill climbing is a heuristic search technique for solving certain mathematical optimization problems in the field of artificial intelligence. In this technique, starting with a suboptimal solution is compared to starting from the base of the hill, and improving the solution is compared to walking up the hill. The optimal solution of the hill climbing technique can be achieved in polynomial time and is an NP-complete problem in which the numbers of local maxima can lead to an exponential increase in computational time. To address these problems, the proposed hill climbing algorithm based on the local optimal solution is applied to the message passing interface, which is a library of routines that can be used to create parallel programs by using commonly available operating system services to create parallel processes and exchange information among these processes. Experimental results show that parallel hill climbing outperforms sequential methods.**

***Keywords*—Hill climbing; heuristic search; parallel processing; Message Passing Interface (MPI)**

I. Introduction

Hill climbing algorithm based on the local optimal solution was proposed and applied to the Message Passing Interface (MPI), which is a library of routines that can be used to create parallel programs in C, C++, and Fortran 77 by using commonly available operating system services to create parallel processes and exchange information among these processes [1]. In this algorithm, the 10 closest points around the current point are scanned, and the cost needed to go from the current point to the next point is obtained by calculating the sum of the obstacles between the current point and the 10 other points. The MPI method is used to validate the performance of the hill climbing algorithm by using parallel and distributed computing systems compared with sequential methods [2].

Hill climbing is a heuristic search technique for solving certain mathematical optimization problems in the field of artificial intelligence [3]. In this technique, starting with a suboptimal solution is compared to starting from the base of the hill, and improving the solution is compared to walking up the hill. The solution is improved repeatedly until a certain condition is maximized and becomes optimal. This technique is mainly used to solve difficult problems computationally [4].

Heuristic search is an artificial intelligence search technique and a computer simulation of thinking that utilizes heuristic for its moves [5, 6]. Heuristic search is a search process that uses domain knowledge in heuristic rules or procedures to direct the progress of a search algorithm, is utilized to prune the search space, and is adopted in applications where a combinatorial explosion indicates that an exhaustive search is impossible [7].

The objective of heuristic search is to produce a solution in a reasonable time frame that is sufficient to solve the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution, but it is still valuable because finding it does not require a prohibitively long time. For large and complex problems, finding an optimal solution path can take a long time and a suboptimal solution that can be obtained rapidly may be useful. Various techniques for modifying a heuristic search algorithm, such as hill climbing, to allow a tradeoff between solution quality and search time have been investigated [8, 9].

The optimal solution of Hill Climbing technique can be achieved in polynomial time and it is one of the NP-Complete problem that the numbers of local maxima can be the cause of exponential computational time. To address these problems parallel and distributed computing systems can be applied to Hill Climbing algorithm. Parallel and distributed computing systems are high-performance computing systems that spread out a single application over many multi-core and multi-processor computers in order to rapidly complete the task. Parallel and distributed computing systems divide large problems into smaller sub-problems and assign each of them to different processors in a typically distributed system running concurrently in parallel. MPI are one these computing systems [10, 11].

The MPI is a standardized means of exchanging messages among multiple computers running a parallel program across a distributed memory. The MPI is generally considered to be the industry standard and forms the basis for most communication interfaces adopted by parallel computing programmers. The MPI is used to improve scalability, performance, multi-core and cluster support, and interoperation with other applications [12].

The rest of the paper is organized as follows. Section II reviews works that are closely related to the hill climbing algorithm. Sections III and IV present the methodology of the new proposed algorithm and its analysis. Section V presents the experimental results. Section VI provides the conclusion.

II. Related Work

Mathematicians and research scientists have found many applications that use heuristic search. The increased use of heuristic search in a wide variety of applications, such as science, engineering, economics, and technology, is due to the advent of personal and large-scale computers.

Rashid and Tao [13] presented an optimized hill climbing algorithm called parallel iterated local search with efficiently accelerated GPUs. They also tested the algorithm by using a typical case study of the graph bisection in computational science. The proposed algorithm minimizes the data transfer between two components to achieve the best performance. Then, the purpose of the parallelism control is to control the generation of the neighborhood for meeting the memory constraints and finding efficient mappings between neighborhood candidate solutions and GPU threads. The authors found through experiments that GPU computing not only accelerates the search process but also exploits parallelism to improve the quality of the obtained solutions for combinatorial optimization problem.

Jiang et al. [14] proposed an optimal power allocation (OPA) method to exert the maximum efficiency of parallel grid-connected inverters. They established the power model of every inverter and compared each model by using the equal power allocation (EPA) method. Theoretically, high overall system efficiency can be achieved using the OPA method. Then, the authors calculated the overall system efficiency with easily measurable electric parameters and realized online optimization by adopting an existing method, such as the hill climbing method. They tested the effectiveness of the proposed method by comparing it with the equivalent power allocation method. They found that the overall system efficiency of the OPA method is higher than that of the EPA method. Moreover, the system using the hill climbing method performs effectively in the dynamic process with short response time.

Kim et al. [15] performed component sizing of power sources of parallel hybrid vehicle by applying the golden section search and hill climbing algorithms. The golden section search algorithm was used in selecting a reduction gear ratio that connects the transmission to the electric motor by using the hill climbing search algorithm to find the optimal engine and electric motor sizes. The use of the hill climbing search algorithm reduces the number of simulations and simultaneously optimizes the capacity of the power source and the gear ratio of the torque coupler. The authors verified the validity of the component sizing results by comparing the global optimal solution obtained by the conventional technique with the solution obtained by the proposed optimization technique.

Robinson et al. [16] presented an improved algorithm for approximating the TSP on fully connected, symmetric graphs by utilizing the GPU. They improved an existing 2-opt hill climbing algorithm with random restarts by considering multiple updates to the current path found in parallel. Their approach has a k-swap function, which allows k number of updates per iteration. The authors showed that their modifications result in a substantial speedup without a reduction in the quality of the result by applying the k-swap method. Their experimental results showed that common assumptions in obtaining good performance for the GPU are not always true, such as saturating the GPU with blocks. Instead, for problems in which the search space can be deterministically enumerated, the number of active blocks can be limited as determined by the hardware. This property allows for reduced memory allocation. A limited amount of memory can be used because each block can allocate the amount of memory upfront.

Chen et al. [17] proposed an automatic machine learning (AutoML) modeling architecture called Autostacker, which is a machine learning system with an innovative architecture for automatic modeling and a well-behaved efficient search algorithm. Autostacker improves the prediction accuracy of machine learning baselines by utilizing an innovative hierarchical stacking architecture and an efficient parameter search algorithm. Neither prior domain knowledge about the data nor feature preprocessing is needed. The authors reduced the time of AutoML by using a naturally inspired algorithm called PHC. They demonstrated the operation and performance of their system by comparing it with human initial trails and related state-of-the-art techniques. They also confirmed the scaling and parallelization ability of their system. The authors also automated the machine learning modeling process by providing an efficient, flexible, and well-behaved system. This system can be generalized into complicated problems and integrated with data and feature processing modules.

III. Methodology

Hill climbing is a heuristic search technique for solving certain mathematical optimization problems in the field of artificial intelligence [18]. In this technique, starting with a suboptimal solution is compared to starting from the base of the hill, and improving the solution is compared to walking up the hill; the solution is improved repeatedly until some condition is maximized and becomes optimal, as illustrated in Fig. 1. This technique is mainly used for solving difficult problems computationally. It focuses on the current and immediate future states and does not maintain a search tree, thereby making it memory efficient [19].

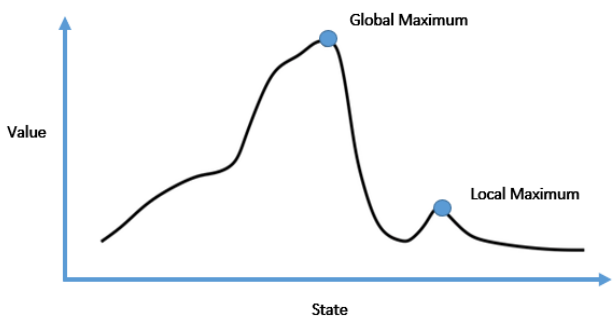

Fig. 1. Hill Climbing.

Hill climbing technique can be used to solve many problems, such as network flow, traveling salesman, and 8-Queens, in which the current state allows for an accurate evaluation function [20]. This technique does not suffer from space-related issues because it focuses on the current state in which previously explored paths are not stored; nonetheless, an optimal solution can be achieved in polynomial time. However, for NP-complete problems, computational time can be exponential based on the number of local maxima [21].

Hill climbing technique comprises several phases, which start by constructing a suboptimal solution considering the constraints of the problem, followed by improving the solution by step and enhancing the solution until no more improvement is possible [22]. This technique is performed following the steps below.

1) Define the current state as initial state.

2) Loop until the goal state is achieved or no more operators can be applied on the current state.

a) Apply an operation to the current state and obtain a new state.

b) Compare the new state with the goal state.

c) Quit if the goal state is achieved.

d) Evaluate the new state with heuristic function and compare it with the current state.

e) If the newer state is closer to the goal than the current state, then update the current state.

In the hill climbing algorithm, achieving the goal is equivalent to reaching the top of the hill.

In this research, a new hill climbing algorithm is proposed on the basis of local optimal solution. In this algorithm, the closest 10 points around the current point are scanned, and the cost needed to go from the current point to the next point is obtained by calculating the sum of the obstacles between the current point and the 10 other points. In this proposed algorithm we have designed the optimization technique as explained in the following equation:

$$Optimize\ Min\left(\sum_{Right\ Paths=1}^{5} w_s * S + w_g * G + w_o * O, \sum_{Left\ Paths=1}^{5} w_s * S + w_g * G + w_o * O\right)$$

Where *Ws* is the weight of the slop, S is the slop of the point, *Sg* is the weight of the gravity, *G* is the gravity at that point, *Wo* is the weight of the obstacles, *O* is the value of the obstacles.

These values will be counted for each path from right and from left. Then, the path which has the lowest cost will be selected.

The scanning approach is performed as follows:

In Table I, the current point is in red (23); each cell has a weight represented by a number. The scanning area is five paths to the right and five paths to the left.

Paths from left:

1) 23→16→16→12→16 (83).
2) 23→15→23→10→32 (103).
3) 23→15→16→56→20 (103).
4) 23→15→16→16→15 (85).
5) 23→15→16→16→23 (93).

Paths from Right:

1) 23→65→26→15→23 (152).
2) 23→23→15→20→32 (113).
3) 23→23→29→30→10 (115).
4) 23→23→29→28→32 (135).
5) 23→23→29→28→72 (175).

Thereafter, we decide which path should be taken depending on the minimum value among the total obstacle weights in each path. In the example above, the best path is number 1 because it has the minimum total value. Fig. 2 illustrates the Proposed Sequential Algorithm.

The above pseudocode is for sequential execution. To parallelize this algorithm, we must follow the following approach:

1) There will be a master node that will generate the matrix.

2) The master node must fill the matrix with the obstacle's weights based on the following equation so that the algorithm can calculate the cost.

$$Cost = w_s * S + w_g * G + w_o * O$$

W_s is the weight of the slop, S is the slop of the point, S_g is the weight of the gravity, G is the gravity at that point, W_o is the weight of the obstacles, O is the value of the obstacles.

1) The master node will broadcast the matrix to all other nodes so they can work in parallel.

2) Each node will calculate its start region from bottom and its end region from top as shown in Fig. 3.

3) All the node will start working at the same time.

4) After they all finish, each node will send the best path for the master node.

5) Finally, the master node will decide which path is the best path based on what did it get from the other nodes.

TABLE I. PROPOSED ALGORITHM EXAMPLE

2	10	20	26	23	23	15	16	10	20	23
51	32	30	15	65	32	16	23	56	15	65
32	23	12	54	72	28	29	30	21	16	72
16	32	15	32	64	23	95	65	12	32	64
2	10	20	26	23	23	15	16	10	20	23
51	32	30	15	65	32	16	23	56	15	65
54	72	28	29	23	23	15	16	16	23	23
32	64	23	95	65	32	16	23	29	30	65

```
Let Mat [100][100];
Set StartPoint;
Let Prow = 0;
Let Check = 0;
Sub GO(ByVal rowIndex As Integer, ByVal Colindex As
Integer)
            If PRow = rowIndex Then
        check = 0
        If rowIndex >= 4 Then
           GO(rowIndex - 1, Colindex + 1)
                    End If
        Exit Sub
      End If
      If PRow = rowIndex Then
         check += 1
      End If
      PRow = rowIndex
      Dim arr As New ArrayList
      Dim Rounds As Integer = 4
      Dim TotalRounds As Integer = 4
      For j As Integer = Colindex To Colindex + 4
         Dim counter As Integer = 0
         Dim sum As Double = 0
         For i As Integer = rowIndex To rowIndex - Rounds Step -
1
            sum += grd.Rows(i).Cells(j + counter).Value
            counter += 1
         Next
         For i As Integer = j - 1 To Colindex Step -1
            If j <> Colindex Then
               sum += grd.Rows(rowIndex).Cells(i).Value
            End If
         Next
         arr.Add(rowIndex - Rounds & "," & Colindex +
TotalRounds)

         arr.Add(sum)
         Rounds -= 1
      Next
      Rounds = 4
      For j As Integer = Colindex To Colindex - 4 Step -1
         Dim counter As Integer = 0
         Dim sum As Double = 0
         For i As Integer = rowIndex To rowIndex - Rounds Step -1
            sum += grd.Rows(i).Cells(j - counter).Value
            counter += 1

         Next
         For i As Integer = j + 1 To Colindex
            If j <> Colindex Then
               sum += grd.Rows(rowIndex).Cells(i).Value
            End If
         Next
         arr.Add(rowIndex - Rounds & "," & Colindex - TotalRounds)
         arr.Add(sum)
         Rounds -= 1
      Next
      Dim min As Integer = arr(1)
      Dim Row As Integer = 0
      Dim Col As Integer = 0
      Dim Sign As Integer = txtDest.Text - Colindex
      Dim Right As Integer = Math.Abs(txtDest.Text - (Colindex
+ TotalRounds))
      Dim Left As Integer = Math.Abs(txtDest.Text - (Colindex -
TotalRounds))
      Dim SelectedPath As Integer = 0
      If Right < Left AndAlso Colindex + TotalRounds <
Colindex + TotalRounds + Right Then
         SelectedPath = Colindex + TotalRounds

      ElseIf Left < Right AndAlso Colindex - TotalRounds >
Colindex - TotalRounds - Left Then
         SelectedPath = Colindex - TotalRounds
         min = arr(11)
      End If
      For i As Integer = 1 To arr.Count - 1 Step 2
         If txtSpiciifc.Text = 1 AndAlso SelectedPath <> 0 Then
            If arr(i) <= min AndAlso arr(i - 1).ToString.Split(",")(1) =
SelectedPath Then
               min = arr(i)
               Row = arr(i - 1).ToString.Split(",")(0)
               Col = arr(i - 1).ToString.Split(",")(1)
            End If
         Else
            If arr(i) <= min Then
               min = arr(i)
               Row = arr(i - 1).ToString.Split(",")(0)
               Col = arr(i - 1).ToString.Split(",")(1)
            End If
         End If

      Next

      grd.Rows(Row).Cells(Col).Style.BackColor = Color.Red
      coloring(Col, Colindex, Row, rowIndex)
      Dim endT As TimeSpan = Now.TimeOfDay
      TotalTime += (endT - start).Milliseconds
      If Row - 4 >= 0 AndAlso Col + 4 < grd.Columns.Count
AndAlso Row > 0 Then
         GO(Row, Col)


            End If
      check = Not check
   End Sub
```

Fig. 2. Sequential Pseudocode.

For example, after broadcasting the matrix to all node, each processor will choose start region and end region depending on its ID. Thus, we will divide the very last row between the processor based on the following equations to get the start region for each processor:

$$Block\ Size = \frac{Number\ of\ starting\ points}{Number\ of\ processors}$$

$$Starts\ From = Processor\ ID * Block\ Size$$

$$Ends\ At = Starts\ From + Block\ Size - 1$$

To get the end region for each processor, we will divide the very first row in the matrix between the processors based on the following equations:

$$Block\ Size = \frac{Number\ of\ Ending\ points}{Number\ of\ processors}$$

$$Starts\ From = Processor\ ID * Block\ Size$$

$$Ends\ At = Starts\ From + Block\ Size - 1$$

Fig. 3 illustrates this example with 10 points and 5 processors.

In the proposed system we have considered three approaches for finding the best path.

Approach #1: From All points below to unknown point above. In this approach the algorithm will start from each point below the hill and try to find a path depending on the discussed algorithm above, but the destination is not specified. So, the algorithm will suggest the path and will determine the destination point. To parallelize this approach each processor will start from the points in its region only.

Approach #2: From All points below to a specific point above. In this approach the algorithm will start from each point below the hill and try to find a path to a specific point above. To parallelize this approach each processor will start from the points in its region only.

Approach #3: From One point below to all points above. In this approach the algorithm will start from a specific point below the hill and try to find a path to all point above the hill. To parallelize this approach each processor will start from the specified point then it will use the End region to determine its destination based on the end region points only.

The evaluation of Hill Climbing technique used only at the current state, it does not suffer from computational space issues, where the source of its computational complexity arises from the time required to explore the problem space. The optimal solution of Hill Climbing technique can be achieved in polynomial time and it is one of the NP-Complete problem that the numbers of local maxima can be the cause of exponential computational time [23]. To address these problems proposed algorithm are applied on message passing interface (MPI) parallel and distributed computing systems with high-performance computing that spread out a single application over many multi-core and multi-processor computers to rapidly complete the task. MPI divide large problems into smaller sub-problems and assign each of them to different processors in a typically distributed system running concurrently in parallel.

In this research, Proposed Hill Climbing techniques are tested on two methods. First method is sequential that accessed code by a single thread. This means that a single thread can only do code in a specific order, hence it being sequential. Second method is MPI that is a library of routines that can be used to create parallel programs in C, C++, and Fortran77 using commonly available operating system services to create parallel processes and exchange information among these processes, as shown in Fig. 4.

The design process of MPI includes vendors (such as IBM, Intel, TMC, Cray, and Convex), parallel library authors (involved in the development of PVM, and Linda), and application specialists. The final version for the draft standard became available in May of 1994 [8].

MPI is a standardized means of exchanging messages among multiple computers running a parallel program across a distributed memory to improve scalability, performance, multi-core and cluster support, and interoperation with other applications. These programs cannot use any MPI communication routine. The two basic routines are MPI_Send, to send a message to another process, and MPI_Recv, to receive a message from another process.

End Regions									
Processor 1		Processor 2		Processor 3		Processor 4		Processor 5	
23	68	789	45	21	32	12	32	32	1
213	32	12	3	5	65	6	56	5	45
1	23	56	6	65	48	78	48	65	12
33	321	12	32	15	35	82	25	25	22
65	65	84	654	987	12	3	123	45	8
23	321	45	94	3	321	123	978	56	23
798	23	546	32	15	32	56	12	32	12
32	32	12	3	15	32	12	31	32	32
Processor 1		Processor 2		Processor 3		Processor 4		Processor 5	
Start Regions									

Fig. 3. Example of Parallelizing the Matrix.

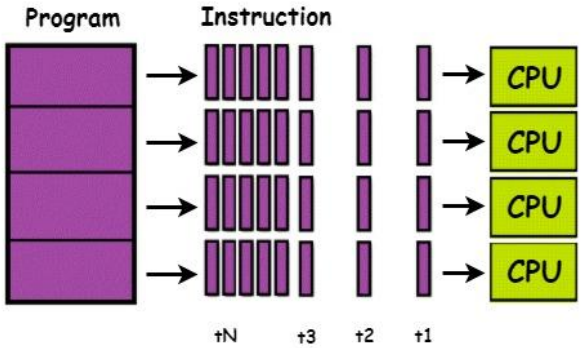

Fig. 4. MPI Parallel Processes.

Proposed algorithm run MPI code in IMAN1, Jordan's first and fastest high-performance Computing resource, funded by JAEC and SESAME. It is available for use by academia and industry in Jordan and the region. In our project, we worked in a Zaina server, an Intel Xeon-based computing cluster with 1G Ethernet interconnection as shown in Table II. The cluster is mainly used for code development, code porting, and synchrotron radiation application purposes. In addition, this cluster is composed of two Dell PowerEdge R710 and five HP ProLiant DL140 G3 servers.

TABLE II. ZAINA TECHNICAL DETAILS

Server	7 Servers (Two Dell PowerEdge R710 and five HP ProLiant DL140 G3)
CPU per server	Dell (2 X 8 cores Intel Xeon) HP (2 X 4 cores Intel Xeon)
RAM per server	Dell (16 GB) HP (6 GB)
Total storage (TB)	1 TB NFS Share
OS	Scientific Linux 6.4

IV. PROPOSED HILL CLIMBING ALGORITHM ANALYSIS

In this section, analysis of the proposed hill climbing algorithm were discussed. Variables that included in all the analysis equation are giving as follows:

Let N = Number of rows in matrix;

Let Paths = Number of scanned paths each time;

Let Points = Number of points in each path;

First; sequential analysis for best paths and parallel analysis for best paths indicate that the proposed algorithm is cost optimal based on the following equation:

A. Sequential Analysis for Best Paths

$$TS = \frac{N}{Points} * (Paths * Points) = N * Paths \quad (1)$$

B. Parallel Analysis for Best Paths

Let P = Number of Processors

$$Tp = TComm + TComp \quad (2)$$

$$TComm = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{\frac{N}{Points}*(Paths*Points)}{P}\right) = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*(Paths)}{P}\right) \quad (3)$$

$$TComp = \left(\frac{N*(Paths)}{P}\right) \quad (4)$$

$$Tp = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*(Paths)}{P}\right) + \left(\frac{N*(Paths)}{P}\right) \quad (5)$$

C. Total Parallel Overhead

$$T = P(t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*(Paths)}{P}\right) + \left(\frac{N*(Paths)}{P}\right)) - (N * Paths) = t_s(N) + t_w(N * Paths) \quad (6)$$

$$Speedup = \frac{N*Paths}{t_s\left(\frac{N}{P}\right)+t_w\left(\frac{N*(Paths)}{P}\right)+\left(\frac{N*(Paths)}{P}\right)} \quad (7)$$

$$Efficiency = \frac{N*Paths}{t_s(N)+t_w(N*Paths)+(N*Paths)} \quad (8)$$

$$Cost = P(t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*(Paths)}{P}\right) + \left(\frac{N*(Paths)}{P}\right) \quad (9)$$

Second; sequential analysis for all to one or one to all and parallel analysis for all to one or one to all indicate that the proposed algorithm is cost optimal based on the following equation.

D. Sequential Analysis for All to One or One to All

$$TS = \frac{N}{Points} * (\frac{Paths}{2} * Points) = N * \frac{Paths}{2} \quad (10)$$

E. Parallel Analysis for All to One or One to All

Let P = Number of Processors

$$Tp = TComm + TComp \quad (11)$$

$$TComm = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{\frac{N}{Points}*\left(\frac{Paths}{2}*Points\right)}{P}\right) = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) \quad (12)$$

$$TComp = \left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) \quad (13)$$

$$Tp = t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) + \left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) \quad (14)$$

F. Total Parallel Overhead

$$T = P(t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) + \left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right)) - \left(N * \frac{Paths}{2}\right) = t_s(N) + t_w\left(N * \frac{Paths}{2}\right) \quad (15)$$

$$Speedup = \frac{N*\frac{Paths}{2}}{t_s\left(\frac{N}{P}\right)+t_w\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right)+\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right)} \quad (16)$$

$$Efficiency = \frac{N*\frac{Paths}{2}}{t_s(N)+t_w\left(N*\frac{Paths}{2}\right)+\left(N*\frac{Paths}{2}\right)} \quad (17)$$

$$Cost = P(t_s\left(\frac{N}{P}\right) + t_w\left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right) + \left(\frac{N*\left(\frac{Paths}{2}\right)}{P}\right)) \quad (18)$$

V. EXPERIMENTS AND RESULTS

This research uses different matrix sizes that contain points that need to go from the current point to the next point in a certain matrix. The cost of moving from the current point to the next point is calculated using the sum of the obstacles between the current point and all the 10 other points. The path is decided depending on the minimum value among the total obstacle weights in each path. The proposed algorithm is tested in sequential and parallel forms coded by MPI. The results are compared in terms of efficiency and speedup ratio.

First, simple hill climbing is used to calculate the time needed to find all the best paths from a specific point to another point with the least cost from that start point. Second, the proposed hill climbing algorithm is utilized to calculate the time required to find the best path from all the points below the

matrix to a specific point above the matrix. Finally, the proposed hill climbing algorithm is adopted to calculate the time needed to find the best path from a specific point below the matrix to all the points above the matrix.

The sequential results of the proposed hill climbing algorithm are tested with various matrix sizes. The algorithm is written in C++. The experimental results are calculated with the 1 core in MPI as shown in Table III.

The MPI results of the proposed hill climbing algorithm are tested using different numbers of cores and matrices. The results are effective and efficient when the number of cores is increased due to the large size of problems that need a high degree of parallelism. Table IV and Fig. 5 show the results for all the best paths. Fig. 6 illustrates all points below a point above, and Fig. 7 presents a point below all points above.

The comparison between MPI results and sequential methods indicates that MPI is always faster and more efficient than sequential methods for different matrix sizes.

Table V presents the calculation results for speedup ratio. Fig. 8 shows the comparison of speedup ratio for all best path results. Fig. 9 reveals the speedup ratio for a point below to all points above. Fig. 10 illustrates the speedup ratio for all points below to a point above.

TABLE III. SEQUENTIAL RUN TIME RESULTS

One CPU	From All Points Below to Unknown Point Above	From All Points Below to One Point Above	From One Point Below to All Points Above
100 * 100	3.200 s	3.150 s	3.300 s
500 * 500	130.230 s	132.230 s	134.230 s
1000 * 1000	593.320 s	598.120 s	601.300 s

TABLE IV. MPI RUN TIME RESULTS

2 CPUs	From All Points Below to Unknown Point Above	From All Points Below to One Point Above	From One Point Below to All Points Above
100 * 100	2.910	2.890	3.210
500 * 500	105.600	106.900	106.230
1000 * 1000	342.250	341.530	342.680
4 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	2.32	2.13	2.23
500 * 500	39.23	38.32	39.65
1000 * 1000	160.2	158.7	158.32
8 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	1.51	1.35	1.56
500 * 500	18.5	18.23	18.9
1000 * 1000	76.45	75.32	76.81
16 CPUs	**From All Points Below to**	**From All Points Below to**	**From One Point Below to All**

	Unknown Point Above	One Point Above	Points Above
100 * 100	1.12	1.23	1.11
500 * 500	12.23	13.89	12.56
1000 * 1000	39.56	38.56	39.15
32 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.927	0.978	0.968
500 * 500	6.2	6.3	6.51
1000 * 1000	19.65	20.3	20.3
64 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.748	0.789	0.868
500 * 500	3.5	3.6	3.78
1000 * 1000	11.2	11.9	12.3
100 CPU	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.592	0.512	0.54
500 * 500	1.9	1.8	1.7
1000 * 1000	6.2	6.9	6.5

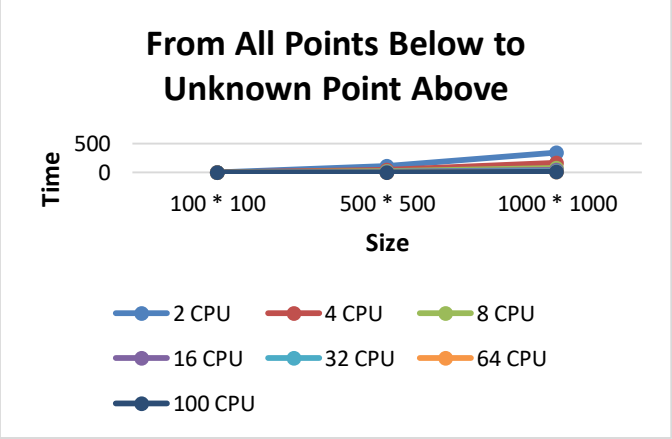

Fig. 5. From All Points below to unknown Point above.

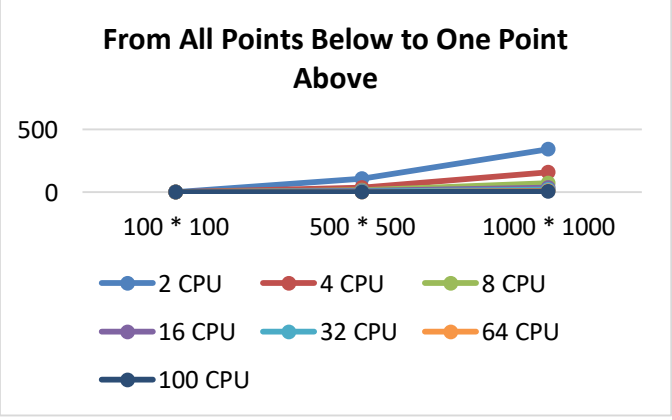

Fig. 6. From All Points below to One Point above Plotting.

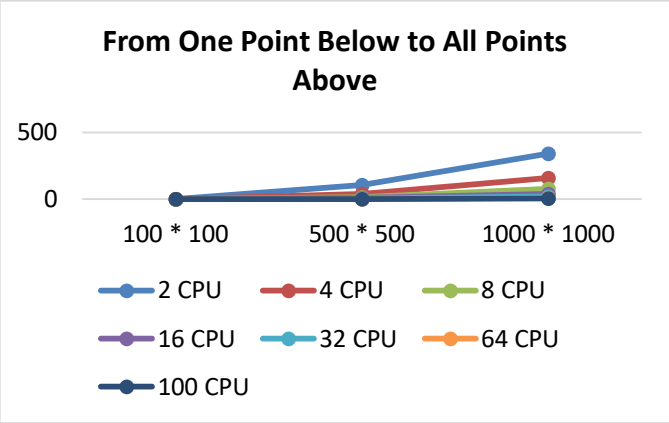

Fig. 7. From One Point below to All Points above Plotting.

TABLE V. SPEEDUP RESULTS

2 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	1.100	1.090	1.028
500 * 500	1.233	1.237	1.264
1000 * 1000	1.734	1.751	1.755
4 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	1.379	1.479	1.480
500 * 500	3.320	3.451	3.385
1000 * 1000	3.704	3.769	3.798
8 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	2.119	2.333	2.115
500 * 500	7.039	7.253	7.102
1000 * 1000	7.761	7.941	7.828
16 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	2.857	2.561	2.973
500 * 500	10.648	9.520	10.687
1000 * 1000	14.998	15.511	15.359
32 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	3.452	3.221	3.409
500 * 500	21.005	20.989	20.619
1000 * 1000	30.194	29.464	29.621
64 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	4.278	3.992	3.802
500 * 500	37.209	36.731	35.511
1000 * 1000	52.975	50.262	48.886
100 CPU	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	5.405	6.152	6.111
500 * 500	68.542	73.461	78.959
1000 * 1000	95.697	86.684	92.508

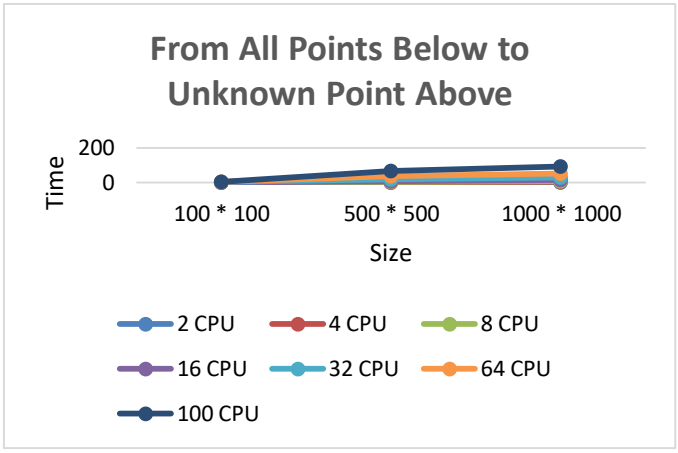

Fig. 8. From All Points below to unknown Point above Speedup Plotting.

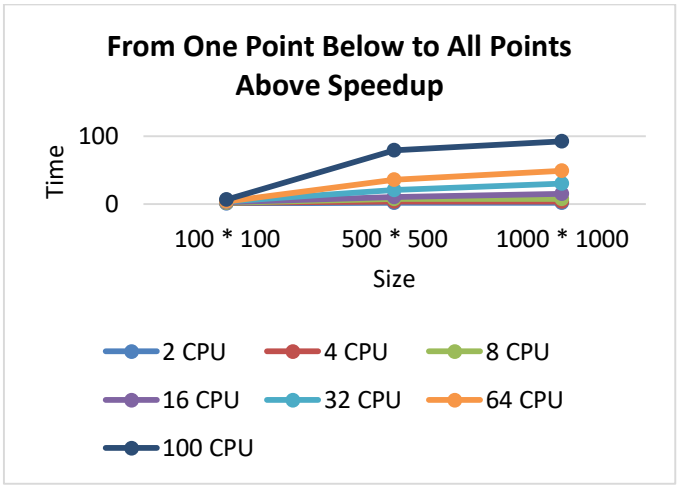

Fig. 9. From One Point below to All Points above Speedup Plotting.

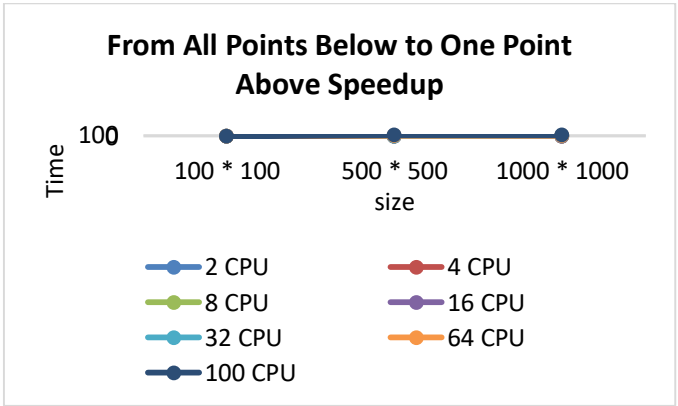

Fig. 10. From All Points below to One Point above Speedup Plotting.

Table VI shows the calculation results for parallel efficiency. Fig. 11 presents the comparison of parallel efficiency for all best path results. Fig. 12 reveals the parallel efficiency for all points below to unknown point above. Fig. 13 illustrates the parallel efficiency for a point below to all points above.

TABLE VI. EFFICIENCY RESULTS

2 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.550	0.545	0.514
500 * 500	0.617	0.618	0.632
1000 * 1000	0.867	0.876	0.877
4 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.345	0.370	0.370
500 * 500	0.830	0.863	0.846
1000 * 1000	0.926	0.942	0.950
8 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.265	0.292	0.264
500 * 500	0.880	0.907	0.888
1000 * 1000	0.970	0.993	0.979
16 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.179	0.160	0.186
500 * 500	0.666	0.595	0.668
1000 * 1000	0.937	0.969	0.960
32 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.108	0.101	0.107
500 * 500	0.656	0.656	0.644
1000 * 1000	0.944	0.921	0.926
64 CPUs	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.067	0.062	0.059
500 * 500	0.581	0.574	0.555
1000 * 1000	0.828	0.785	0.764
100 CPU	**From All Points Below to Unknown Point Above**	**From All Points Below to One Point Above**	**From One Point Below to All Points Above**
100 * 100	0.054	0.062	0.061
500 * 500	0.685	0.735	0.790
1000 * 1000	0.957	0.867	0.925

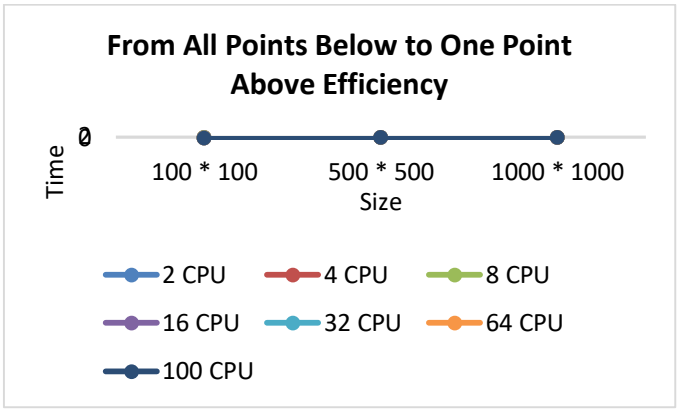

Fig. 11. From All Points below to One Point above Efficiency Plotting.

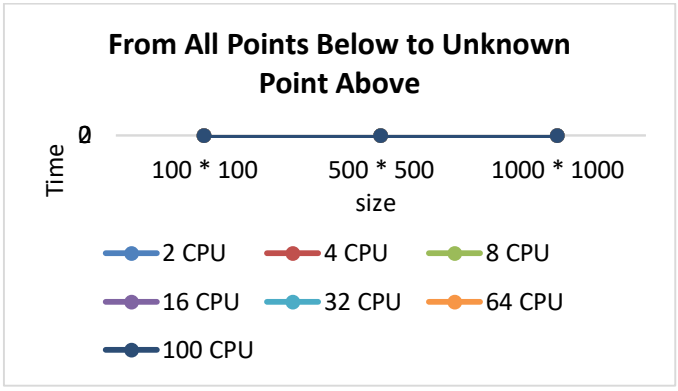

Fig. 12. From All Points below to unknown Point above Efficiency Plotting.

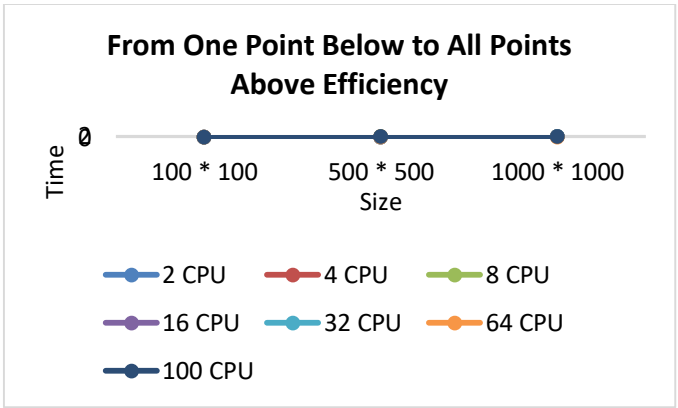

Fig. 13. From One Point below to All Points above Efficiency Plotting.

VI. CONCLUSION

Heuristic search is a search process that uses domain knowledge in heuristic rules or procedures to direct the progress of a search algorithm. Hill climbing is a heuristic search technique for solving certain mathematical optimization problems in the field of artificial intelligence. In this technique, starting with a suboptimal solution is compared to starting from the base of the hill, and improving the solution is compared to walking up the hill. The optimal solution of the hill climbing technique can be achieved in polynomial time and is an NP-complete problem in which the numbers of local maxima can lead to an exponential increase in computational time. To address these problems, the proposed hill climbing algorithm

based on the local optimal solution is applied to the message passing interface, which is a library of routines that can be used to create parallel programs by using commonly available operating system services to create parallel processes and exchange information among these processes. Experimental results show that parallel hill climbing outperforms sequential methods.

This research uses different matrix sizes that contain points that need to go from the current point to the next point in a certain matrix. The cost of moving from the current point to the next point is calculated using the sum of the obstacles between the current point and all the 10 other points. The path is decided depending on the minimum value among the total obstacle weights in each path. The proposed algorithm is tested in sequential and parallel forms coded by MPI. The results are compared in terms of efficiency and speedup ratio.

The comparison between MPI results and sequential methods indicates that MPI is always faster and more efficient than sequential methods for different matrix sizes. Fig. 7, 8, and 9 show the comparison results for sizes 100×100, 500×500, and 1000×1000, respectively. The MPI outperforms the sequential methods; thus, the research goal is achieved.

REFERENCES

[1] Snir, M., Otto, S., Huss-Lederman, S., Dongarra, J., & Walker, D. (1998). MPI--the Complete Reference: The MPI core (Vol. 1). MIT press.

[2] Chira, C., Horvath, D., & Dumitrescu, D. (2011). Hill-Climbing search and diversification within an evolutionary approach to protein structure prediction. BioData mining, 4(1), 23.

[3] Selman, B., & Gomes, C. P. (2006). Hill-climbing search. Encyclopedia of Cognitive Science, 81, 82.

[4] Cook, C. M., Rosenfeld, A., & Aronson, A. R. (1976). Grammatical inference by hill climbing. Information Sciences: an International Journal, 10(2), 59-80.

[5] Apter, M. J. (1970). The Computer Simulation of behaviour. London: Hutchinson. Allgemeinverständliche, inzwischen etwas veraltete Darstellung der Simulationsmethodik mit Diskussion von Anwendungen aus Bereichen des Lernens, des Problemlösens, des Mustererkennens, der Sprache und der Persönlichkeitstheorie bis hin zum Problem des Bewußtseins.

[6] Masadeh, R., Mahafzah, B. A., & Sharieh, A. (2019). Sea Lion optimization algorithm. International Journal of Advanced Computer Science and Applications, 10(5), 388-395.

[7] Goswami, S., Das, A. K., Guha, P., Tarafdar, A., Chakraborty, S., Chakrabarti, A., & Chakraborty, B. (2017). An approach of feature selection using graph-theoretic heuristic and hill climbing. Pattern Analysis and Applications, 1-17.

[8] Hansen, E. A., & Zhou, R. (2007). Anytime heuristic search. Journal of Artificial Intelligence Research, 28, 267-297.

[9] Masadeh, R., Sharieh, A., & Sliet, A. (2017). Grey wolf optimization applied to the maximum flow problem. International Journal of Advanced and Applied Sciences, 4(7), 95-100.

[10] Fox, Geoffrey C., Steve W. Otto, and Anthony JG Hey. "Matrix algorithms on a hypercube I: Matrix multiplication." Parallel computing 4.1 (1987): 17-31.

[11] Masadeh, R., Alzaqebah, A., Smadi, B., Masadeh, E. (2020). Parallel Whale Optimization Algorithm for Maximum Flow Problem. Modern Applied Science, 14(3), 30-44.

[12] Gropp, W. D., Gropp, W., Lusk, E., & Skjellum, A. (1999). Using MPI: portable parallel programming with the message-passing interface (Vol. 1). MIT press.

[13] Rashid, M. H., & Tao, L. (2017, October). Parallel Combinatorial Optimization Heuristics with GPUs. In Computer Science and Intelligent Controls (ISCSIC), 2017 International Symposium on (pp. 118-123). IEEE.

[14] Jiang, W., Wang, P., Wang, J., & Wang, L. (2017). Optimal power allocation for parallel grid-connected inverters based on lagrangian function method. Chinese Journal of Electrical Engineering, 3(3), 68-76.

[15] Kim, J., Kim, G., & Park, Y. I. (2018). Component Sizing of Parallel Hybrid Electric Vehicle Using Optimal Search Algorithm. International Journal of Automotive Technology, 19(4), 743-749.

[16] Jiang, W., Wang, P., Wang, J., & Wang, L. (2017). Optimal power allocation for parallel grid-connected inverters based on lagrangian function method. Chinese Journal of Electrical Engineering, 3(3), 68-76.

[17] Chen, B., Mo, W., Chattopadhyay, I., & Lipson, H. (2018). Autostacker: an Automatic Evolutionary Hierarchical Machine Learning System.

[18] Mincu, R. S., & Popa, A. (2018, July). Heuristic Algorithms for the Min-Max Edge 2-Coloring Problem. In International Computing and Combinatorics Conference (pp. 662-674). Springer, Cham.

[19] Harman, M., & McMinn, P. (2007, July). A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation. In Proceedings of the 2007 international symposium on Software testing and analysis (pp. 73-83). ACM.

[20] Gámez, J. A., Mateo, J. L., & Puerta, J. M. (2011). Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. Data Mining and Knowledge Discovery, 22(1-2), 106-148.

[21] Khari, M., & Kumar, P. (2017). Empirical Evaluation of Hill Climbing Algorithm. International Journal of Applied Metaheuristic Computing (IJAMC), 8(4), 27-40.

[22] Chan, W. K. V., D'Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., & Page, E. A Global and Local Search Approach to Quay Crane Scheduling Problem.

[23] Nicolau, M., & McDermo, J. (2017). Late-Acceptance and Step-Counting Hill-Climbing GP for Anomaly Detection.