Intelligent system design using hyper-heuristics

Nelishia Pillay

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, South Africa

ABSTRACT

Determining the most appropriate search method or artificial intelligence technique to solve a problem is not always evident and usually requires implementation of different approaches to ascertain this. In some instances a single approach may not be sufficient and hybridization of methods may be needed to find a solution. This process can be time consuming. The paper proposes the use of hyper-heuristics as a means of identifying which method or combination of approaches is needed to solve a problem. The research presented forms part of a larger initiative aimed at using hyper-heuristics to develop intelligent hybrid systems. As an initial step in this direction, this paper investigates this for classical artificial intelligence uninformed and informed search methods, namely depth first search, breadth first search, best first search, hill-climbing and the A* algorithm. The hyper-heuristic determines the search or combination of searches to use to solve the problem. An evolutionary algorithm hyper-heuristic is implemented for this purpose and its performance is evaluated in solving the 8-Puzzle, Towers of Hanoi and Blocks World problems. The hyper-heuristic employs a generational evolutionary algorithm which iteratively refines an initial population. On each iteration the evolutionary algorithm uses tournament selection to select parents. The mutation and crossover operators are applied to the chosen parents to produce offspring for the next generation. The hyper-heuristic was able to identify a search or combination of searches to produce solutions for the twenty 8-Puzzle, five Towers of Hanoi and five Blocks World problems. Furthermore, admissible solutions were produced for all problem instances. A different search or combination of searches was evolved for each problem instance. This study has highlighted the potential of hyper-heuristics for the automated design of intelligent systems. Given this success, future work will investigate the use of hyper-heuristics for the design of intelligent hybrid systems for high-level reasoning, which will combine genetic algorithms, tabu search, variable neighbourhood search and simulated annealing. The automated design of intelligent systems has long term benefits for the software industry as a means of reducing the man hours needed for system design.

KEYWORDS: Hyper-heuristics, evolutionary algorithms, algorithm design

CATEGORIES: I.2 [**Artificial Intelligence**]

ARTICLE HISTORY

Received 24 August 2014
Accepted 23 March 2015

1 INTRODUCTION

The process of identifying the most suitable artificial intelligence technique to use to solve a problem can be time consuming. This decision is usually based on past applications and successes of the methods and similarities of the problem to previous applications for which the method was effective. This often involves implementing the different approaches that are identified as having the potential to solve the problem and comparing the performance to determine the method that will produce the best results. In some instances a single approach may not be sufficient and hybridization of methods is needed to solve the problem. The research presented in this paper forms part of a larger initiative aimed at using hyper-heuristics [1] to develop intelligent hybrid systems.

The hyper-heuristic will select or combine different artificial intelligence techniques to solve the problem at hand. The paper investigates the effectiveness of this approach for traditional artificial intelligence searches, namely depth first, breadth first, best first, hill-climbing and the A* algorithm, as tested on the 8-Puzzle, Towers of Hanoi and Blocks World problems.

Hyper-heuristics aim to produce a more generalized solution over a problem domain, rather than producing the best results for some problem instances as is typical of traditional optimization methods [1]. Hyper-heuristics have proven to be very effective in solving combinatorial optimization problems such as timetabling, cutting and packing problems, Boolean satisfiability and vehicle routing problems, amongst

Email: Nelishia Pillay pillayn32@ukzn.ac.za

others. More recently attempts have also been made to employ hyper-heuristics to generalize over domains [2].

Hyper-heuristics select or combine low-level heuristics by exploring a heuristic space [1]. The low-level heuristics can be constructive or perturbative. Constructive heuristics are used to create a solution to a problem, while perturbative heuristics improve an initial solution created randomly by using a construction heuristic. Examples of constructive heuristics include heuristics to choose the next event to schedule in timetabling and heuristics to select the bin to place an item in for packing problems. A commonly used low-level perturbative heuristic is the swap operator which swaps rows or columns in a timetable or items amongst bins for packing problems.

Hyper-heuristics are hence categorized as selection constructive, selection perturbative, generation constructive and generation perturbative. Selection hyper-heuristics select the heuristic to apply next when constructing or improving a solution. Generation hyper-heuristics create new low-level heuristics by combining low-level heuristics or components of these heuristics. Genetic programming [3, 4] has been primarily used by generative hyper-heuristics to create new heuristics. Methods employed by selection constructive hyper-heuristics include tabu-search, variable neigbourhood search, simulated annealing and evolutionary algorithms. Selection perturbative hyper-heuristics perform single point or multi-point search. Single point search hyper-heuristics are comprised of a heuristic selection and move acceptance component while multi-point hyper-heuristics employ a population based method such as evolutionary algorithms or particle swarm optimization to select low-level heuristics.

This study investigates the use of a multi-point search selection perturbative hyper-heuristic to identify the appropriate search or combination of searches to use to solve the problem at hand. An evolutionary algorithm is used to explore the heuristic space. The low-level perturbative heuristics are the searches, i.e.,

- depth first,
- breadth first,
- best first,
- hill-climbing, and
- A* algorithm.

The performance of the hyper-heuristic is evaluated on three classical artificial intelligence problems that uninformed and informed searches are typically used to solve, namely:

- the 8-Puzzle problem,
- the Towers of Hanoi problem, and
- the Blocks World problem.

The following section provides an overview of the traditional artificial intelligence searches. Section 3 describes the hyper-heuristic for artificial intelligence (HHAI). The experimental setup used to evaluate the hyper-heuristic is presented in Section 4, and Section 5 discusses the performance of HHAI in solving twenty 8-Puzzle, five Towers of Hanoi and five Blocks World problem instances. The paper concludes with a summary of the findings and details of future work in Section 6.

2 ARTIFICIAL INTELLIGENCE SEARCHES

Traditional artificial intelligence searches are categorized as uninformed and informed searches [5]. Commonly used uninformed searches include depth first and breadth first searches. The depth first search traverses an entire branch and backtracks when the end of the branch is reached. The algorithm for the depth first search presented in [5] is depicted in Fig. 1.

```
def dfs (in Start, out State)
  open = [Start]
  closed = []
  State = failure
  while (open <> []) AND (State <> success)
    begin
    remove the leftmost state from open,
        call it X
    if X is the goal, then
      State = success
    else begin
      generate children of X
      put X on closed
      eliminate the children of X on open
          or closed
      put remaining children on left end
          of open
    end else
  endwhile
  return State
end def
```

Figure 1: Depth-first search [5]

```
def bfs (in Start, out State)
 open = [Start]
 closed = []
 State = failure
 while (open <> []) AND (State <> success)
   begin
    remove the leftmost state from open,
        call it X
    if X is the goal, then
      State = success
    else begin
      generate children of X
      put X on closed
      eliminate the children of X on open
          or closed
      put remaining children on right end
          of open
    end else
 endwhile
 return State
enddef
```

Figure 2: Breadth first search [5]

The depth first search is not an admissible search and

```
def bestfs(in Start, out State)
  open = [Start]
  close = []
  State = failure
  while (open <> []) AND (State <> success)
    begin
      remove the leftmost state from open, call it X
      if X is the goal, then
          State = success
        else begin
        generate children of X
        for each child of X do
        case
          the child is not on open or closed:
            begin
              assign the child a heuristic value
              add the child to open,
            end
              the child is already on open:
            if the child was reached by a shorter path then
              give the state on open the shorter path
          the child is already on closed:
            if the child is reached by a shorter path then
            begin
              remove the state from closed
              add the child to open
            end
          endcase
          endfor
          put X on closed
          re-order states on open by heuristic merit (best leftmost)
    endwhile
  return State
end
```

Figure 3: Best-first search [5]

is hence not guaranteed to find the minimum cost solution path. Whereas the depth first search searches vertically, the breadth first search explores the space horizontally. All the nodes at a depth are expanded before moving to the next depth, from top to bottom.

Fig. 2 illustrates the breadth first search presented in [5]. The breadth first search is an admissible search that is guaranteed to find the minimum cost solution path.

Informed searches use heuristics to guide the search, thereby possibly reducing the space that needs to be explored. The best first search and hill-climbing are informed searches. The best first search searches globally and hill-climbing locally. The best first search algorithm is listed in Fig. 3 and hill-climbing in Fig. 4. Both searches use a heuristic $h(n)$, which is problem dependent, to choose which state to explore next. This heuristic is an estimate of the cost from the node n to the goal state. The best first search and hill-climbing are not admissible algorithms. The A algorithm is essentially the best first search which in addition to the heuristic $h(n)$ uses an estimate of the cost of getting from the start state to the node n. This is defined as $f(n) = h(n) + g(n)$, where $h(n)$ is the estimated cost of getting from the state n to the goal state and $g(n)$ which estimates the cost of getting from the start state to the state n. In this study $g(n)$ is the depth at which n is in the search tree. Section 4 defines $h(n)$ for each of the problem domains HHAI is tested on. If $h(n)$ is an admissible heuristic (i.e. if it always underestimates the cost of getting from n to the goal state), the A algorithm is admissible (i.e. it produces the minimum cost solution path) and it is referred to as the A* algorithm.

3 HYPER-HEURISTIC FOR INTELLIGENT SYSTEM DESIGN

This section describes the evolutionary algorithm hyper-heuristic used to design artificial intelligence search algorithms. This study improves on the initial attempt at this reported in [6]. The mutation operator used in [6] was not very effective and this study redefines this operator and also introduces a crossover operator for regeneration. In the study in [6] the evolutionary algorithm terminated when a heuristic combination producing a solution was found. However, as a result of this, the hybridized search produced was not admissible, i.e. the optimal solution path with a minimum number of moves was not necessarily found. In this study this is remedied by extending the ter-

```
def hill_climbing(in Start, out State)
  open = [Start], close = []
  State = failure
  while (open <> []) AND (State <> success)
    begin
      remove the leftmost state from open, call it X
      if X is the goal, then
        State = success
      else begin
        generate children of X
        for each child of X do
        case
          the child is not on open or closed
            begin
              assign the child a heuristic value
            end
          the child is already on open
            if the child was reached by a shorter path then
              give the state on open the shorter path
          the child is already on closed:
            if the child is reached by a shorter path then
              begin
                remove the state from closed
              end
        endcase
        endfor
      put X on closed
      re-order the children states by heuristic merit (best leftmost)
      place the reordered list on the leftmost side of open
    endwhile
  return State
end
```

Figure 4: Hill-climbing search [5]

mination criteria to include producing an admissible hybridized search. The fitness function has also been defined to cater for two objectives, namely, producing a solution and the minimum solution path.

The evolutionary algorithm implemented is the generational algorithm depicted in Fig. 5. The algorithm begins by creating an initial population of heuristic combinations of low-level heuristics.

```
Create initial population
Repeat
  Evaluate the population
  Select parents
  Perform regeneration using mutation
    and crossover
Until the termination criterion is met
```

Figure 5: Generational evolutionary algorithm

The low-level heuristics are the depth first, breadth first, best first, hill-climbing and A/A* algorithm searches and the number of iterations of each search to be performed. Each combination is evaluated by using it to solve the problem instance. Tournament selection is used to select parents for regeneration during which the mutation and crossover operators are used to create the offspring of each generation. These processes are described in the sections that follow.

3.1 Initial population generation

Each element of the population, i.e. chromosome, is a string comprised of characters representing each of the searches as listed below and the number of iterations of each search to be performed:

- Depth first search (d)
- Breadth first search (b)
- Best first search (s)
- Hill-climbing (h)
- A/A* search (a)

Thus each gene is of the form *iterations;search*, e.g. *10;s* which represents ten iterations of the best first search. The search for each gene is determined by randomly selecting a character representing the searches. In the study presented in [6] the number of iterations was chosen to be in the range of 1 and a specified maximum which is problem dependent. However, it was found in this study that adding the preset maximum as an offset to the randomly chosen number of iterations in the range 1 to the maximum is more effective. An example of a chromosome *is 19;a,12;b,8;s*. The algorithm represented by this chromosome performs the A algorithm for 19 iterations, followed by the breadth first search for 12 iterations and the best first search for 8 iterations.

```
Parent 1: 72;b,53;s    Parent 2: 13;h,20;b,22;d,12;s
Crossover point: 2     Crossover point: 3
Offspring 1: 72;b,22;d,12,s
Offspring 2: 13;h,20;b, 53;s
```

Figure 6: Crossover example

3.2 Evaluation and selection

Each chromosome is evaluated by using it to create a solution to the problem instance. Pareto fitness, which assesses the chromosome for two objectives—namely, correctness of solution and optimal cost solution path—is used for fitness evaluation. The heuristic $h(n)$, which estimates the cost of the state n from the goal state, is used as a measure of solution correctness. The length of the solution path is used to measure the second objective. Tournament selection is used to choose the parents of each generation. In the case of mutation the tournament selection method is evoked once to obtain a parent while in the case of crossover it is called twice as the operator is applied to two parents. The tournament selection method returns the fittest element of a tournament of t individuals as a parent. Each element of the tournament is randomly selected from the population and selection is with replacement, i.e. an individual can be selected more than once as a parent. In comparing the fitness of chromosomes, solution correctness is given priority. The chromosome with a lower value of $h(n)$ is considered to be fitter. If two chromosomes have the same value for correctness, the chromosome with the lower path length is treated as fitter.

A solution algorithm or an admissible solution algorithm may not use all the genes. For example if the chromosome is *20;a,12;b,8;s*, an admissible solution may be found after performing twenty iterations of the A* algorithm and six iterations of the breadth first search. In this case the chromosome is pruned and the solution algorithm is *20;a,6;b*.

3.3 Regeneration

The mutation and crossover operators are used to create the offspring of each generation.

The mutation operator firstly randomly chooses a mutation point in the parent. The mutation operator used in [6] replaced the gene at the mutation point with a new gene. This was not very effective and the mutation operator was redesigned. This operator performs one of the following five operations which is randomly chosen:

Gene deletion. Delete the gene at the mutation point if the size of the chromosome is greater than one.

Search replacement. Replace the search in the gene at the mutation point with a randomly selected search. The number of iterations remains unchanged.

Iteration replacement. Replace the number of iterations in the gene, using the same approach used to determine the number of iterations for each gene for initial population generation. The search in the gene remains unchanged.

Gene insertion. Insert a new gene before the mutation point.

Gene replacement. Replace the gene at the mutation point with a new gene, as in [6].

The crossover operator is applied to copies of two parents chosen using tournament selection with Pareto fitness. Crossover points are randomly selected in both the parents and the chromosome substrings are swapped at these points to produce two new offspring. The process is illustrated in Fig. 6.

Previous research on evolutionary algorithm hyper-heuristics shows that returning the fitter of both offspring instead of two offspring is more effective [7, 8]. Thus, the fitter offspring is produced as the result of the crossover operator in this study.

4 EXPERIMENTAL SETUP

The hyper-heuristic was evaluated on three different domains, namely, the 8-Puzzle problem, Towers of Hanoi and the Blocks World problem. HHAI was implemented in Java using JDK 1.7.0 and all simulations were run on a Windows 7 machine with an Intel Core i7 processor. The following sections describe each of the problems, the problem sets used for each problem and the HHAI parameter values for each problem.

4.1 8-Puzzle problem

The 8-Puzzle problem involves moving tiles on a 3×3 board to get from an initial state to a goal state. The board contains 8 numbered tiles and a space. An example is illustrated in Fig. 7.

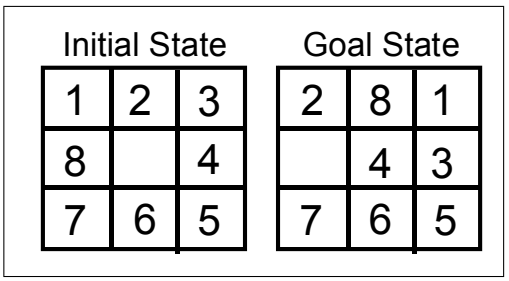

Figure 7: 8-Puzzle problem example

A search method takes the initial and goal states as input and outputs a list of moves to get from the initial to the goal state. The moves are defined in terms of moving the space rather than moving the tiles. The space can be moved up, down, left and

right. The twenty 8-Puzzle problem instances in Table 1, of differing difficulty, from past studies and online assignments were used to evaluate the HHAI.

Trial runs were conducted to empirically determine the parameter values of HHAI to be used in solving the 8-puzzle problem. These values are listed in Table 2.

Table 1: 8-Puzzle problem instances

	Initial state	Goal state	Known optimum (minimum moves)
1	123804765	134862705	5
2	123804765	281043765	9
3	123804765	281463075	12
4	134805726	123804765	6
5	231708654	123804765	14
6	231804765	123804765	16
7	123804765	231804765	16
8	283104765	123804765	4
9	876105234	123804765	28
10	867254301	123456780	31
11	647850321	123456780	31
12	123804765	567408321	30
13	806547231	012345678	30
14	641302758	012345678	14
15	158327064	012345678	12
16	328451670	012345678	12
17	035428617	012345678	10
18	725310648	012345678	15
19	412087635	123456780	17
20	162573048	123456780	10

Table 2: Parameter values for the 8-puzzle problem

Parameter	Value
Population size	500
Max. no. of generations	50
Max. length	10
Tournament size	4
Max. no. of iterations	10
Crossover rate	50%
Mutation rate	50%

Heuristics that have generally been used by informed searches for the 8-Puzzle problem include the number of tiles out of place, tile reversal and the Manhattan distance or a combination as these [5]. The Manhattan distance has proven to be the most effective [9] and hence is used in this study. This heuristic is essentially the sum of the distances of each tile from its position in the goal state. This heuristic is also used by HHAI as a fitness measure (see Section 3.2).

4.2 Towers of Hanoi

An instance of the Towers of Hanoi problem with three discs is illustrated in Fig. 8. Solving the problem requires moving the discs from the leftmost pole to the rightmost pole, without at any time placing a larger ring on a smaller ring. Only one ring can be moved at a time.

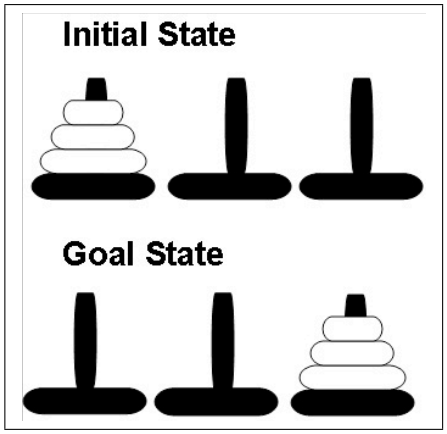

Figure 8: An example of the Towers of Hanoi problem

Table 3: Towers of Hanoi problem instances

Problem	Known optimum (minimum moves)
3 discs	7
4 discs	15
5 discs	31
6 discs	63
7 discs	127

Table 4: Parameter values for the Towers of Hanoi problem

Parameter	Value
Population size	500
Max. no. of generations	50
Max. length	10
Tournament size	4
Max. no. of iterations	20 (3, 4 and 5 discs), 100 (6 discs), 200 (7 discs)
Crossover rate	50%
Mutation rate	50%

The HHAI was tested on five instances of the problem with 3, 4, 5, 6 and 7 discs. The details are listed in Table 3.

As with the 8-Puzzle problem, the parameter values for HHAI were determined empirically by performing trial runs. These are listed in Table 4.

The following heuristic is used by the informed searches and as the fitness measure by HHAI:

$$\begin{aligned}\text{heuristic} = {} & 2 \times \text{number of discs on Pole 1} \\ & + 2 \times \text{number of discs on Pole 2} \\ & + \text{sum of the distance of each disc on Pole 3} \\ & \quad \text{from its position in the goal state}\end{aligned}$$

4.3 Blocks World

The Blocks World problem is illustrated in Fig. 9. The initial state contains blocks on a table, some of which form a stack. Solving the problem involves identifying the moves needed for the blocks to be stacked as indicated in the goal state. The legal moves include moving a block from the table to the top of the stack and moving the topmost block on the stack to the table.

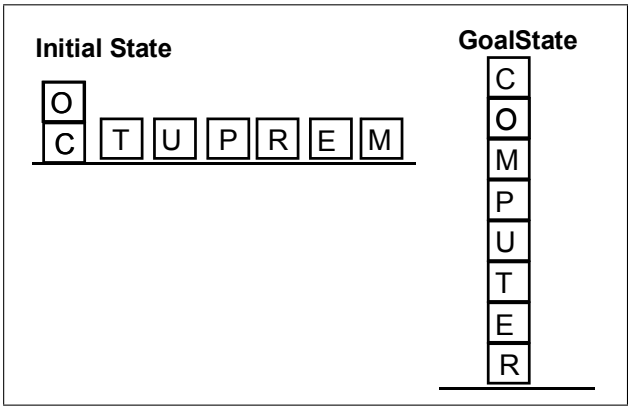

Figure 9: An example of the Blocks World problem

Instances of the Blocks World problem were created by choosing five words of different lengths. The initial state for each word was created by randomly determining whether the block corresponding to each letter is on the stack or table. The goal state in each case is the blocks correctly stacked to spell out the word. The instances are listed in Table 5.

Table 5: Informed searches for Blocks World

	Value	Initial state		Known optimum (minimum moves)
		Stack	Table	
BW1	computer	ope	cmutr	11
BW2	artificial	aiic	rtfial	14
BW3	intelligence	nelige	itlnce	16
BW4	translators	anlators	Trs	7
BW5	program-ming	prgmng	orami	13

The parameter values used by HHAI for this problem domain, which were determined by performing trial runs, are listed in Table 6.

Table 6: Parameter values for Blocks World problem

Parameter	Value
Population size	500
Max. no. of generations	50
Max. length	10
Tournament size	4
Max. no. of iterations	50
Crossover rate	50%
Mutation rate	50%

The following is used as a heuristic for the informed searches and the fitness measure by HHAI:

$$\begin{aligned}\text{heuristic} = {} & \text{number of blocks missing from the stack} \\ & + \text{number of incorrectly-placed blocks} \\ & \text{on the stack}\end{aligned}$$

5 RESULTS AND DISCUSSION

This section discusses the performance of the HHAI in designing hybridized search algorithms to solve the three classical artificial intelligence problems. The first subsection applies the traditional searches to these problems to evaluate how these searches work independently, providing a benchmark against which the performance of the hybridized searches produced by HHAI can be compared. This also provides a means of determining the level of difficulty of each of the problem instances for the three domains. The sections that follow report on the performance of the HHAI for the 8-Puzzle, Towers of Hanoi and Blocks World problems respectively. Section 5.5 compares the results for the 8-Puzzle and Towers of Hanoi problems to that of the initial attempt in [6]. A summary of the results is presented in Section 5.6. The searches and HHAI are evaluated in terms of whether they are able to induce a set of moves to produce a solution and whether the number of moves is the optimal, i.e. the minimum number of moves[1]. Thus a smaller value for the solution path length (number of moves) is indicative of a better solution.

5.1 Performance of traditional searches

This section reports on the performance of the depth first search (DFS), breadth first search (BFS), best first search, hill-climbing and A* algorithm, in solving problem instances of the 8-Puzzle, Towers of Hanoi and Blocks World problems. Each search was given a maximum of thirty minutes within which to find a solution. This maximum was chosen based on trial runs performed with uninformed and informed searches. The maximum runtimes of these trials were approximately thirty seconds. It was decided to use thirty minutes to give the searches a fair chance of finding a solution without the overhead of high runtimes. The performance of the searches are depicted in Table 7 to Table 12. For each search the number of moves, i.e. length of the solution path, and the runtime is displayed. A hyphen "—" indicates that the search was not able to find a solution within the time limit. The heuristics used by the informed searches for each of the problem domains are discussed in section 4. From Table 7 and Table 8 it is evident that puzzles 9 to 13 are the most difficult and could only be solved by heuristic searches. All three heuristic searches were able to find solutions for all problem instances. The A* algorithm has produced optimal results, with slightly higher runtimes for the more difficult problem instances.

[1]The known minimum number of moves for each problem instance is provided in Section 4.

Table 7: Performance of uninformed searches in solving the 8-Puzzle problem (A = number of moves, B = runtime in seconds)

Problem	DFS		BFS	
	A	B	A	B
1	5	< 1	5	< 1
2	9	< 1	9	1
3	12	< 1	12	1
4	6	< 1	6	< 1
5	14	2	14	2
6	16	6	16	6
7	16	6	16	6
8	4	< 1	4	< 1
9	—	< 1	—	< 1
10	—	< 1	—	< 1
11	—	< 1	—	< 1
12	—	< 1	—	< 1
13	—	< 1	—	< 1
14	14	2	14	2
15	12	1	12	1
16	12	1	12	1
17	10	< 1	10	1
18	15	4	15	4
19	17	12	17	11
20	10	1	10	< 1

Table 8: Performance of informed searches in solving the 8-Puzzle problem (A = number of moves, B = runtime in seconds)

Problem	Best first		Hill climbing		A* algorithm	
	A	B	A	B	A	B
1	5	< 1	5	< 1	5	< 1
2	31	< 1	13	< 1	9	< 1
3	34	< 1	16	< 1	12	1
4	6	< 1	6	< 1	6	< 1
5	22	< 1	22	< 1	14	< 1
6	16	1	20	< 1	16	< 1
7	16	< 1	16	< 1	16	< 1
8	4	< 1	4	< 1	4	< 1
9	58	< 1	48	< 1	28	9
10	47	< 1	433	< 1	31	29
11	47	< 1	109	< 1	31	28
12	64	< 1	126	< 1	30	1
13	47	< 1	103	< 1	31	28
14	24	< 1	72	< 1	14	< 1
15	36	< 1	42	< 1	12	< 1
16	38	< 1	120	< 1	12	< 1
17	10	1	48	< 1	10	< 1
18	15	< 1	53	< 1	15	1
19	47	< 1	221	< 1	17	< 1
20	12	< 1	306	< 1	10	< 1

Table 9: Performance of uninformed searches in solving the Towers of Hanoi problem (A = number of moves, B = runtime in seconds)

Problem	DFS		BFS	
	A	B	A	B
3 discs	9	1	7	1
4 discs	29	< 1	15	< 1
5 discs	81	< 1	31	< 1
6 discs	245	< 1	63	1
7 discs	729	< 1	127	1

Table 10: Performance of informed searches in solving the Towers of Hanoi problem (A = number of moves, B = runtime in seconds)

Problem	Best first		Hill-climbing		A* algorithm	
	A	B	A	B	A	B
3 discs	11	1	15	< 1	7	< 1
4 discs	19	< 1	35	< 1	15	< 1
5 discs	39	< 1	93	< 1	31	< 1
6 discs	95	< 1	257	1	63	1
7 discs	243	< 1	747	1	127	1

Table 11: Performance of uninformed searches in solving the Blocks World problem (A = number of moves, B = runtime in seconds)

Problem	DFS		BFS	
	A	B	A	B
BW1	—		—	
BW2	—		—	
BW3	—		—	
BW4	191	1	7	2
BW5	—		—	

Table 12: Performance of informed searches in solving the Blocks World problem (A = number of moves, B = runtime in seconds)

Problem	Best first		Hill-climbing		A* algorithm	
	A	B	A	B	A	B
BW1	11	< 1	11	< 1	11	1
BW2	14	< 1	14	< 1	14	1
BW3	16	< 1	16	< 1	16	7
BW4	7	< 1	7	< 1	7	< 1
BW5	13	< 1	13	< 1	13	1

Table 9 and Table 10 display the results of the uninformed and informed searches respectively in solving the Towers of Hanoi problem. All the searches were able to produce solutions for all problem instances, with the breadth first search and A* algorithms producing optimal solutions.

It can be seen from the results in Table 11 that the uninformed searches were only able to solve one problem instance of the Blocks World problem within the time limit of thirty minutes. Heuristic search is needed to find a solution for the four problem instances. All three searches were able to produce optimal solutions.

5.2 Performance of HHAI in solving the 8-Puzzle problem

The HHAI was able to find solutions for all twenty problem instances. Due to the stochastic nature of evolutionary algorithms, thirty runs were performed for each problem. Table 13 lists the success rate, i.e. percentage of runs producing a solution, admissibility rate, i.e. percentage of runs producing a solution with the minimum number of moves, the minimum and maximum path length of the solutions produced in the thirty runs.

Table 13: Performance of HHAI in solving the 8-Puzzle problem (A = success rate, B = admissibility rate, C = known minimum, D = minimum solution path length, E = maximum solution path length)

	A	B	C	D	E
1	100%	100%	5	5	5
2	100%	100%	9	9	9
3	100%	100%	12	12	12
4	100%	100%	6	6	6
5	100%	100%	14	14	14
6	100%	100%	16	16	16
7	100%	100%	16	16	16
8	100%	100%	4	4	4
9	100%	3%	28	28	40
10	100%	37%	31	31	39
11	100%	37%	31	31	37
12	100%	100%	30	30	30
13	100%	53%	31	31	39
14	100%	100%	14	14	14
15	100%	100%	12	12	12
16	100%	100%	12	12	12
17	100%	100%	10	10	10
18	100%	100%	15	15	15
19	100%	100%	17	17	17
20	100%	100%	10	10	10

The search algorithms evolved by HHAI for Puzzle 1 produced admissible solutions for all thirty runs with a runtime of less than a second for each run. For five of the runs the algorithm induced was *59;b*. This algorithm performed 59 iterations of the breadth first search to solve the problem. For the remaining twenty five runs, the algorithm evolved was *5;s*. This algorithm performs five iterations of the best first search using the Manhattan distance heuristic.

Similarly, for Puzzle 2 HHAI induced algorithms solving the puzzle with the minimum number of moves for all thirty runs with a runtime of less than a second. For this problem there was more variety in the algorithms producing the best result on each run. An algorithm produced on eleven of the runs was *24;a*, which is 24 iterations of the A* algorithm. Examples of other algorithms induced that produced optimal solutions include *22;a,2;s* and *29;b,5;s*. All the algorithms producing optimal solutions begin with either the A* algorithm or breadth first search.

Admissible solutions were also produced by the induced algorithms on all thirty runs within a second for Puzzle 3. Thirty seven of the solution algorithms began with the A* search and the remaining three algorithms with the breadth first search. Examples of the induced algorithms include *37;a*, *35;a,2;s*, *34;a,11;b*, and *29;b,8;s*.

HHAI has performed similarly on puzzles 4 to 8, 12 and 14 to 20. For all of these puzzles admissible solutions were produced with an average runtime of less than a second for the thirty runs. The algorithm induced on all thirty runs for Puzzle 4 is *6;s*, i.e. six iterations of the best first search using the Manhattan distance heuristic. The induced algorithms solving Puzzle 5 begin with either the A* or breadth first search algorithms, e.g. *26;b,10;s*, *37;a,39;b,29;b*, and *37;a,10;s*. For Puzzle 6 the solution algorithms begin with either the A* algorithm, breadth first search or best search, e.g. *23;a,39;s,31;b*, *45;b,34;s,33;h,31;h* and *36;s,30;s,28;a,22;d*. This algorithms evolved for Puzzle 7 also began with one of these three searches. Some of the algorithms evolved for this puzzle used only the best first search or A* algorithm. All the algorithms using the breadth first search, used it together with the A* or best first search. Essentially two algorithms were evolved for Puzzle 8, namely, *4;s* and *29;b*.

HHAI induced algorithms that produced solutions for all thirty runs for Puzzle 9. The algorithm producing an optimal solution evolved by HHAI is *61;b,60;a,15;s* which combines the breadth first, A* algorithm and the best first search, beginning with the breadth first. Neither of the uninformed searches were able to solve this puzzle. The algorithms induced for this puzzle have a minimum number of moves of 28 and a maximum of 40, with the number of moves for a majority of the algorithms in the range 30 to 36. These algorithms have performed better than both the best first search and hill-climbing which solve this puzzle with 58 and 48 moves respectively. The A* algorithm produces a solution with a minimum number of moves in 9 seconds while the HHAI takes 26 seconds. This is expected as the HHAI evolves a population of 500 hundred algorithms over more than one generation.

The evolved algorithms producing optimal solutions for Puzzle 10, Puzzle 11 and Puzzle 13 begin with either the breadth first search or A* algorithm and combine two or more of the breadth first search, best first search and A* algorithm. The number of runs producing admissible algorithms is not as high as for the other puzzles. It is hypothesized that this

could be improved with further tuning of parameter values.

HHAI induced algorithms producing optimal solutions for all thirty runs for Puzzle 12 and puzzles 14 to 20. Examples of the algorithms are listed in Table 14 for each of the puzzles. For Puzzle 14 *40;b,9;s* was evolved for fifteen of the runs. Similarly, for Puzzle 15 the algorithm *34;b,8;s* was produced on twenty three of the thirty runs. All the algorithms for Puzzle 16 began with the A* algorithm and some of the algorithms were comprised of only the A* algorithm. The algorithm evolved on all runs for Puzzle 17 is *12;s*. For Puzzle 18 the algorithm 38;s was evolved on nineteen of the thirty runs. Algorithms evolved for Puzzle 19 were comprised of either the breadth first search, best first search or the A* algorithm and the best first search, with the breadth first and A* algorithm preceding the best first search. All the algorithms for Puzzle 20 perform the breadth first search or start with the breadth first search followed by the best first search.

5.3 Performance of HHAI in solving the Towers of Hanoi problem

HHAI derived algorithms solving all 5 instances of the Towers of Hanoi problem. As shown in Table 15, all the algorithms were admissible; that is, they produced solutions with the shortest solution path for each of the problem instances.

HHAI is able to evolve admissible algorithms for the problem with 3, 4 and 5 discs within a second, with an average runtime for 6 discs being 18 seconds and 5.42 minutes for 7 discs. Example solutions algorithms are listed in Table 16. The algorithm evolved on all thirty runs for the 3 disc problem is *24;b*, which uses only the breadth first search. The algorithms evolved for the 4 disc problem either applies the breadth first, best first or A* algorithms, or combines two of these searches. Similarly, the algorithms for the 5 disc problem either apply these three searches separately or combine two or three of the searches. All the algorithms generated for the 6 disc problem begin with the breadth first search or A* algorithm. The algorithms are quite varied and combine two to four of all four searches. All of the algorithms evolved for the 7 disc problem except 2 combine the breadth first search and the A* algorithm or apply just the breadth first search. The remaining two algorithms begin with the best first search and combine the best first search and A* algorithm.

5.4 Performance of HHAI in solving the Blocks World problem

HHAI evolved admissible algorithms for all five blocks world instances. The results are listed in Table 17. Admissible algorithms were induced on all thirty runs. The average runtimes for BW1, BW2, BW3, BW4 and BW5 are less than a second, eight seconds, four seconds, less than a second and two seconds respectively. The algorithms produced by HHAI performed better than the uninformed searches and comparably to the informed searches in solving the five problem instances.

Table 18 lists examples of the algorithms evolved for each of the problem instances. For BW1 the algorithms on 28 of the runs implemented the best first search. For two of the runs the algorithm *42;s,20;b* was evolved combining the best first and breadth first search. Similarly, all the algorithms evolved for BW2 except two performed the best first search. Of the remaining two algorithms one combined the breadth first and best first search and the other the A* algorithm and best first search.

Essentially two types of algorithms were evolved for BW3. The first algorithm performed the best first search on twelve runs) and the second combined the breadth first search and best first search (on fifteen runs). An algorithm combining the A* and best first algorithms was produced on two runs and an algorithm performing hill-climbing on one run. For 28 of the runs the algorithm *10;s* was produced for the BW4, and *21;a* for two of the runs. Three algorithms were induced for BW5, namely, *37;b,10;s*, *37;s,10;s* and *37;a,10;s*. All three include the best first search, with one of the algorithms combining the breadth first and best first searches and the remaining algorithm combining the A* algorithm and best first searches.

5.5 Comparison with previous work

This section empirically compares the performance of the hyper-heuristic in [6] to that presented in this study to design search algorithms, comprised of the traditional artificial intelligence searches, for solving the 8-Puzzle and Towers of Hanoi problem. A comparison of performance on the Blocks World problem is not possible as the hyper-heuristic in [6] was not applied to this domain. HHAI was developed to address the shortcomings identified in the previous version reported in [6] (see Section 3). The differences between HHAI and the hyper-heuristic in [6] include:

- A Pareto comparison of fitness including both correctness and optimality of the solution path in HHAI.
- An improved mutation operator which includes gene deletion, iteration replacement, search replacement and gene insertion, in addition to gene replacement used in [6].
- A crossover operator that swaps sections of chromosomes to produce an offspring in HHAI.

Both hyper-heuristics were able to produce optimal solutions for all puzzles. Table 19 lists the admissibility rate for both hyper-heuristics for each of the twenty problem instances. It is evident from Table 19 that HHAI performs better at producing admissible algorithms than the hyper-heuristic in [6]. For Puzzle 9 and Puzzle 12, which the hyper-heuristic in [6] was not able to find admissible algorithms, the best evolved algorithm produced a minimum number of moves of 32 and 38 respectively.

Table 20 lists the admissibility rate for the hyper-heuristic in [6] and HHAI for the Towers of Hanoi problem. Both hyper-heuristics have produced solutions for all instances of this problem.

Table 14: Example optimal algorithms for the 8-Puzzle problem

Puzzle	Example optimal algorithms
12	27;a,26;h 27;s,21;h,33;b 20;b,26;h,2;b
14	40;b,9;s 22;b,10;s 27;a,20;a,8;b
15	34;b,8;s 58;b,6;s 18;a
16	32;a 26;a,5;s 31;a,1;b
17	12;s
18	38;s 34;s,14;b 23;a,11;s
19	35;b,12;s 36;a,12;s
20	40;b,5;s 22;b,28;b

Table 15: Performance of HHAI in solving the Towers of Hanoi problem (A = success rate, B = admissibility rate, C = known minimum, D = minimum solution path length, E = maximum solution path length)

	A	B	C	D	E
3 discs	100%	100%	7	7	7
4 discs	100%	100%	15	15	15
5 discs	100%	100%	31	31	31
6 discs	100%	100%	63	63	63
7 discs	100%	100%	127	127	127

Table 16: Example optimal algorithms for the Towers of Hanoi problem

Puzzle	Example optimal algorithms
3 discs	24;b
4 discs	34;b,12;b 41;a,6;b 46;a 45;s,8;s 44;a,2;s 30;b,8;s
5 discs	36;b,66;a,15;s 43;b,57;b,24;b 32;a,26;a,27;s,24;b
6 discs	195;b,314;b,288;d,226;s 228;a,258;b 211;b,275;b,131;h,208;s
7 discs	834;a,522;b 440;b,544;b,126;b 393;s,422;s,332;a,157;a

Table 17: Performance of HHAI in solving the Blocks World problem (A = success rate, B = admissibility rate, C = known minimum, D = minimum solution path length, E = maximum solution path length)

	A	B	C	D	E
BW1	100%	100%	11	11	11
BW2	100%	100%	14	14	14
BW3	100%	100%	16	16	16
BW4	100%	100%	7	7	7
BW5	100%	100%	13	13	13

Table 18: Example optimal algorithms for the Blocks World problem

Puzzle	Example optimal algorithms
BW1	$44; s, 1; s$ $45; s$ $42; s, 20; b$
BW2	$249; s$ $245; s, 4; s$ $239; b, 10; s$ $239; a, 10; s$
BW3	$49; s, 21; s$ $49; b, 21; s$ $49; a, 21; s$ $16; h$
BW4	$10; s$ $21; a$
BW5	$37; b, 10; s$ $37; s, 10; s$ $37; a, 10; s$

Table 19: Performance comparison for the 8-Puzzle problem

	Hyper-heuristic in [6]	HHAI
1	2%	100%
2	60%	100%
3	70%	100%
4	100%	100%
5	30%	100%
6	40%	100%
7	30%	100%
8	60%	100%
9	0%	3%
10	10%	37%
11	20%	37%
12	0%	100%
13	0%	53%
14	50%	100%
15	90%	100%
16	20%	100%
17	60%	100%
18	90%	100%
19	70%	100%
20	30%	100%

Table 20: Performance comparison for the 8-Puzzle problem

	Hyper-heuristic in [6]	HHAI
1	50%	100%
2	20%	100%
3	0%	100%
4	0%	100%
5	0%	100%

The minimum number of moves that the best algorithm induced by the hyper-heuristic in [6] for the 5 disc, 6 disc and 7 disc problem is 32, 125 and 226 respectively. HHAI has also performed better in terms of admissibility for this domain as well.

5.6 Summary

This section provides a summary of the performance of the searches applied individually to the problem instances and HHAI. The uninformed searches were not able to find solutions for all of the 8-Puzzle problem instances. The difficult problem instances are clearly Puzzle 9 to Puzzle 13. The runtimes for the uninformed searches ranged from less than a second to a maximum of seventeen seconds, the more difficult problem instances taking longer to solve. The informed searches were able to produce solutions for all twenty problem instances with only the A* algorithm inducing admissible solutions. All three searches were able to solve all problems within a second with the exception of the A* algorithm on Puzzle 9, Puzzle 10, Puzzle 11 and Puzzle 13, which took 9, 29, 28 and 29 seconds to solve respectively. This again emphasizes the difficulty of these problem instances. All searches were able to produce solutions for the Towers of Hanoi problem instances within a second. Admissible solutions were generated by the breadth first search and A* algorithm. The uninformed searches were only able to solve one problem instance for the Blocks World problem, namely BW4. All three informed searches induced admissible solutions for all five Blocks World problem instances. All runtimes were less than a second, with an exception of the A* algorithm in solving BW3 which took seven seconds.

Due to the stochastic nature of HHAI thirty runs were performed for each problem instance and the performance of HHAI evaluated over the thirty runs. HHAI was able to produce admissible solutions for all problem instances of the 8-Puzzle problem. Admissible solutions were found on thirty runs for all the problem instances with an exception of Puzzle 9, Puzzle 10, Puzzle 11 and Puzzle 13, with admissible solutions produced on 3%, 37%, 37% and 53% of the thirty runs respectively. HHAI induced admissible solutions for all thirty runs for each problem instance of Towers of Hanoi and Blocks World problems. The runtimes for HHAI ranged from a minimum of less than a second for a majority of the problem instances to a maximum of 26 seconds. HHAI was found to produce more than one admissible solution algorithm, i.e. combination of searches, for a majority of the problem instances. Patterns were found in the evolved algorithms for the different problem instances, e.g. all algorithms beginning with an admissible search.

This study was aimed at improving the initial attempt in [6]. Improvements made include incorporating a measure of optimality of the solution path in the fitness function, a mutation operator which performs gene deletion, iteration replacement, search replacement and gene replacement, and the use of crossover. HHAI outperformed the hyper-heuristic in [6], producing admissible solutions for all instances of the 8-Puzzle and Towers of Hanoi problems.

6 CONCLUSION AND FUTURE WORK

The research presented in this paper forms part of a larger initiative aimed at using hyper-heuristics for the design of intelligent hybrid systems. In an attempt to investigate the potential of hyper-heuristics for this purpose, this study examines the use of an evolutionary algorithm hyper-heuristic (HHAI) to design traditional search algorithms to solve three classical artificial intelligence problems, namely, the 8-Puzzle problem, Towers of Hanoi and Blocks World. HHAI was able to produce solution algorithms for all instances for all three problem domains.

In addition to this, HHAI was able to evolve algorithms producing admissible solutions for all problem instances. For some problem instances the same algorithm was evolved on all thirty runs indicating that only one admissible solution algorithm exists for the problem instance. In some cases although different algorithms were generated on different runs, a pattern

could be found in the algorithms producing optimal solutions, e.g. the algorithms begin with the same search, two or three of the same searches are combined.

Some searches were also found to perform well for certain problem domains. For example, the admissible solution algorithms induced for all instances of the Blocks World problem were comprised essentially of the best first search. The algorithms generated by HHAI were found to outperform the uninformed searches in solving the difficult instances of the 8-Puzzle problem and the Blocks World problem.

Given the success of this study and hence the potential of hyper-heuristics for algorithm design, future work will investigate the use of hyper-heuristics for designing intelligent hybrid systems for high-level reasoning. The low-level heuristics in this case will be genetic algorithms, tabu search, variable neighbourhood search and simulated annealing. The main contribution of this research is the automated design of intelligent systems. This has long term benefits for the software industry as a means of reducing the man hours needed for intelligent system design.

ACKNOWLEDGEMENTS

The author would like to thank the reviewers for their helpful comments and suggestions. This work is based on the research supported in part by the National Research Foundation of South Africa for the grant CSUR13091742778. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF does not accept any liability in this regard.

REFERENCES

[1] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan and R. Qu. "Hyper-heuristics: A survey of the state of the art". *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013. DOI `http://dx.doi.org/10.1057/jors.2013.71`.

[2] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic et al. "Hyflex: A benchmark framework for cross-domain heuristic search". In *Evolutionary computation in combinatorial optimization*, pp. 136–147. Springer, 2012. DOI `http://dx.doi.org/10.1007/978-3-642-29124-1_12`.

[3] J. R. Koza. *Genetic programming: On the programming of computers by means of natural selection*, vol. 1. MIT Press, 1992.

[4] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone. *Genetic programming: An introduction*, vol. 1. Morgan Kaufmann, 1998.

[5] G. F. Luger. *Artificial intelligence: Structures and strategies for complex problem solving*. Pearson Education, 2005.

[6] N. Pillay. "A study of hyper-heuristics for hybridizing search". In *Advances in artificial intelligence—Proceedings of ALEA 2013, 9–12 September, Portugal*, pp. 128–139. 2013.

[7] N. Pillay. "Evolving hyper-heuristics for the uncapacitated examination timetabling problem". *Journal of the Operational Research Society*, vol. 63, no. 1, pp. 47–58, 2012. DOI `http://dx.doi.org/10.1057/jors.2011.12`.

[8] N. Pillay and W. Banzhaf. "A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem". In *Progress in artificial intelligence*, pp. 223–234. Springer, 2007.

[9] A. Reinefeld. "Complete solution of the Eight-Puzzle and the benefit of node-ordering in IDA*". In *Proceedings of the International Joint Conference on Artificial Intelligence, Chambery, Savoi, France (Sept. 1993)*, pp. 248–253. 1993.