2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Genetic Algorithm with Self-Adaptive Mutation Controlled by Chromosome Similarity

Daniel Smullen, Jonathan Gillett, Joseph Heron, Shahryar Rahnamayan

***Abstract*—This paper proposes a novel algorithm for solving combinatorial optimization problems using genetic algorithms (GA) with self-adaptive mutation. We selected the N-Queens problem ($8 \leq N \leq 32$) as our benchmarking test suite, as they are highly multi-modal with huge numbers of global optima. Optimal static mutation probabilities for the traditional GA approach are determined for each N to use as a best-case scenario benchmark in our conducted comparative analysis. Despite an unfair advantage with traditional GA using optimized fixed mutation probabilities, in large problem sizes (where $N > 15$) multi-objective analysis showed the self-adaptive approach yielded a 65% to 584% improvement in the number of distinct solutions generated; the self-adaptive approach also produced the first distinct solution faster than traditional GA with a 1.90% to 70.0% speed improvement. Self-adaptive mutation control is valuable because it adjusts the mutation rate based on the problem characteristics and search process stages accordingly. This is not achievable with an optimal constant mutation probability which remains unchanged during the search process.**

I. Introduction

GENETIC ALGORITHMS (GA) are particularly effective to solve complex or large size combinatorial problems [3], [2], [1]. One well-known problem is the N-Queens problem, which deals with placing queens on a chessboard such that they do not attack one another. An efficient and common approach to tackle these problems (especially for large values of N) are genetic algorithms, by encoding positions of the N-Queens on the chessboard into the chromosome. As with any genetic algorithm, evaluating the fitness of the chromosomes is required to determine the quality of candidate solutions [10], but the N-Queens problem is different in that only those candidates with perfect fitness are considered correct solutions to the problem, which means a zero-attack chessboard (seen in Figure 1).

Tuning up control parameters has a crucial effect on the GA's performance [14], [16], [9]. We have found that by replacing the traditional GA's static mutation probability with a self-adaptive mutation probability, a better performance can be achieved when solving the N-Queens problem by considering different performance perspectives. Our objectives focus on 1) finding as many solutions as possible within a fixed budget, and 2) finding the first solution as quickly as possible. The proposed approach adapts based on the similarity of chromosomes in the population. The increase in performance is compounded in larger versions of the problem where the search space grows extremely large, such as the situation encountered when attempting to solve higher-order N-Queens. In this domain, deterministic methods prove useless. The problem is in P, and the best known deterministic approaches are of $O(N^2)$ [24]. As such, number of combinations of queens that must be placed tentatively onto the board while seeking even the first solution becomes enormous very quickly [22], but our approach prevails.

Our software solution was written in Java, and is available through GitHub at the following URL: http://git.io/RKfAvQ. It is open source licensed under the *GNU General Public License, Version 3*.

Daniel Smullen, Jonathan Gillett, Joseph Heron, and Shahryar Rahnamayan are with The Faculty of Engineering and Applied Science at the University of Ontario Institute of Technology, Oshawa, Ontario, Canada (email: {daniel.smullen, jonathan.gillett, joseph.heron}@uoit.net, shahryar.rahnamayan@uoit.ca).

We would like to thank the SHARCNET organization for graciously providing the high performance computing facilities we needed to run our experiments.

II. Background Review

The N-Queens problem requires that a number of queens, N, are placed on an $N \times N$ chessboard such that they will not attack each other. The N-Queens problem is multi-modal, with many local optima and global optima. Like many other complex optimization problems, the N-Queens problem becomes staggeringly complex as the problem size increases [5]. The problem is bound by the rules of chess.

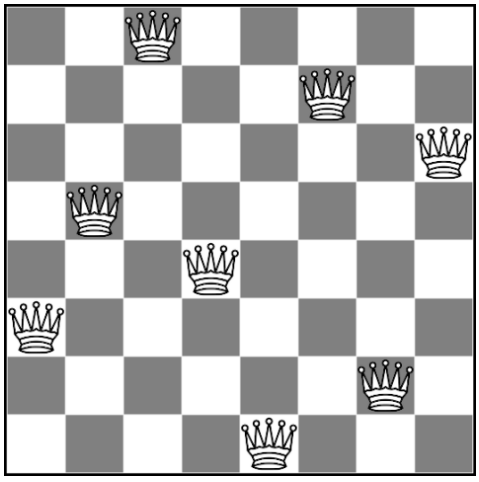

Fig. 1. A Distinct Solution to the 8-Queens Problem (Zero-Attack Chessboard)

Our solution includes protective embedded constraints which restrict the structure of the chromosome - this prevents queens from sharing rows, allowing only for queens to attack

each other on a shared column or the diagonal. When two queens share the same diagonal, a collision occurs for each queen, meaning that the board state is not a solution. A solution is found when no queens on the board can attack another. The search space size is $N!$ when an exhaustive method is applied to find all possible solutions.

TABLE I
N-QUEENS PROBLEM DISTINCT SOLUTIONS

N	Search Space Size ($N!$)	Distinct Solutions [23]
8	40,320	92
9	362,880	352
10	3,628,800	724
11	39,916,800	2,680
12	479,001,600	14,200
13	6,227,020,800	73,712
14	87,178,291,200	365,596
15	$1.307674368 \times 10^{12}$	2,279,184
16	$2.092278989 \times 10^{13}$	14,772,512
17	$3.556874281 \times 10^{14}$	95,815,104
18	$6.402373706 \times 10^{15}$	666,090,624
19	$1.216451004 \times 10^{17}$	4,968,057,848
20	$2.432902008 \times 10^{18}$	39,029,188,884
21	$5.109094217 \times 10^{19}$	314,666,222,712
22	$1.124000728 \times 10^{21}$	2,691,008,701,644
23	$2.585201674 \times 10^{22}$	24,233,937,684,440
24	$6.204484017 \times 10^{23}$	227,514,171,973,736
25	$1.551121004 \times 10^{25}$	2,207,893,435,808,352
26	$4.032914611 \times 10^{26}$	22,317,699,616,364,044
...	...	Unknown

Table I shows the massive exponential increase in problem size as N increases, and the proportionally huge increase in the number of distinct solutions. Based on the scale of the problem at higher values of N, using deterministic methods is futile - the problem is simply too big [21]. For values of N greater than 26, the problem becomes intractable. The number of distinct solutions for large values of N is unknown, and the search space size becomes enormous.

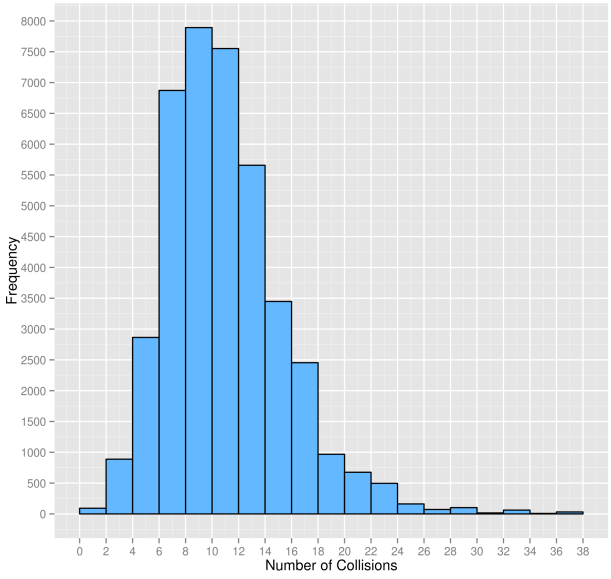

Fig. 2. 8-Queens Problem Collisions Histogram, Showing Distribution Based on Fitness Values

8-Queens is the classical version of N-Queens. The histogram found in Figure 2 shows the full deterministic (brute force) traversal of the 8-Queen problem landscape, which is characteristic of N-Queens. It details the myriad possibilities for the number of board states which can occur even in a relatively small problem size. Not pictured in this figure are the 4 instances of 44 simultaneous collisions, and 2 instances of 56 collisions. These occur when 7 or all 8 queens are all placed on the diagonals, respectively. Figure 2 clearly shows that the problem is highly multi-modal even for small problem sizes ($N = 8$). In this figure, 40228 non-optimal solutions with 92 global optima are displayed, showing their distribution based on their fitness values. The random distribution of solutions across the landscape is not completely uniform and there are many local optima which still represent non-optimal solutions (such as those solutions with only 2 to 4 collisions - better than most, but still not valid boards).

While finding the most solutions given a fixed number of generations was the main objective of our proposed approach, fundamentally there are two variant objectives in tackling N-Queens. Finding the first solution as fast as possible is an alternative goal. The latter problem can be solved in polynomial time through conventional means [23], [24], but finding as many solutions as possible is a challenging target which makes deterministic algorithms impractical.

III. RELATED WORK

Deterministic attempts to solve N-Queens have been successful only for values of N up to 26 [21], [25], and attempts to find single solutions quickly have been made for versions of the problem up to $N = 500,000$ [22]. There are no known integer sequences or proofs which can represent the number of distinct solutions for $N > 26$, nor is there a deterministic means which can find them all within reasonable time frames. Traditional GAs have been used to find multitudinous solutions within fixed time frames [11], [12]. The same approach is utilized in our experiments.

Since tuning control parameters has a profound effect on the performance of GAs, one might conclude that there is an optimal static mutation probability [6] for solving N-Queens using the traditional approach. Hesser and Männer [15] attempted to find such a value for GA in general problems but failed, because they are problem oriented values. Finding the pragmatic best case scenario for fixed mutation probabilities (M_c) within traditional GA proved to be useful for comparison in our experiments, which can be seen in our results.

Other alternative approaches to enhancing GA for combinatorial optimization sought to increase population diversity in order to overcome wasteful convergence to local optima, which is problematic in N-Queens since it has a huge number of local and global optima. This behavior prevents the search from finding more solutions. Combining standard GA with randomly generated individuals into the population is well studied [14], but fails to adapt to suit the problem size. Still, exploring the nature of genetic diversity provides valuable

insight into the motivation for increasing diversity among candidate solutions during the search process dynamically.

Adaptive GAs have historically manipulated control parameters by changing the mutation probability over time [14] such that it is high during exploration or low during exploitation. Other approaches used the overall mean population fitness to adjust genetic operators independent from time [13], [16], [9]. Alternative adaptive approaches have changed mutation probabilities between two values based on fitness values [16], while others attempted to individually specify genetic operator probabilities for each phenotype, encoded with meta-operators that govern their evaluations [7]. Many previous adaptive approaches sought to yield more fruitful explorations of the problem landscape by tuning GA control parameters, but most approaches have been limited by being naïve to either the problem size or the state of exploration and diversity. However, these historical adaptive methodologies yield many advantages over traditional GA in that they preclude the need for *a priori* knowledge about optimal control parameters, mitigating the need to 'tune them up' with protracted test runs prior to long-term evaluation, which saves time.

IV. Proposed Approach

Our main motivation for this new self-adaptive approach was to leverage the observed tendency in nature for mutations to occur as a result of inbreeding with genetically similar specimens [18], [19]. Nature adjusts the mutation rate based on the diversity of the population. Our desire was to replicate this behavior in GA. We noted that mutations that resulted from inbreeding increased genetic diversity in biological organisms, affecting their fitness in nature sometimes for better or for worse. Therefore, it stood to reason that a similar phenomenon might occur in GA. Exploring this idea yielded surprisingly positive results.

Figure 3 shows the flowchart for the self-adaptive mutation approach. Areas where this algorithm differs from traditional GA are highlighted with a darker shade.

A. Chromosome Structure

We used a GA with an integer-encoded chromosome to represent the N-Queens positions on the chessboard for all of our tests independent of their approach. Our chromosome structure includes protective embedded constraints which reduce the size of the search space drastically. These constraints are imposed through our data structure, which consists of a 1-dimensional array (of size N with zero-based numbering) that stores the column position of the queens at each array index. The index is used to represent the row position of each queen, preventing more than one queen from appearing in the same row. Values stored inside the array must be between 0 and $N-1$, preventing a queen from being placed outside the chessboard. However, we do not restrict the values within the array from being repeated, which can occur as a result of the crossover or mutation operators. When repetition occurs within the array, a collision will occur on the respective column. Our implementation uses traditional GA operators - selection, crossover, and mutation. Ordinary roulette wheel selection [20] is used in conjunction with single-value uniform mutation and single-point crossover/recombination [17]. This means that where mutation occurs, genes may take any value within the bounds of the problem size with equal probability.

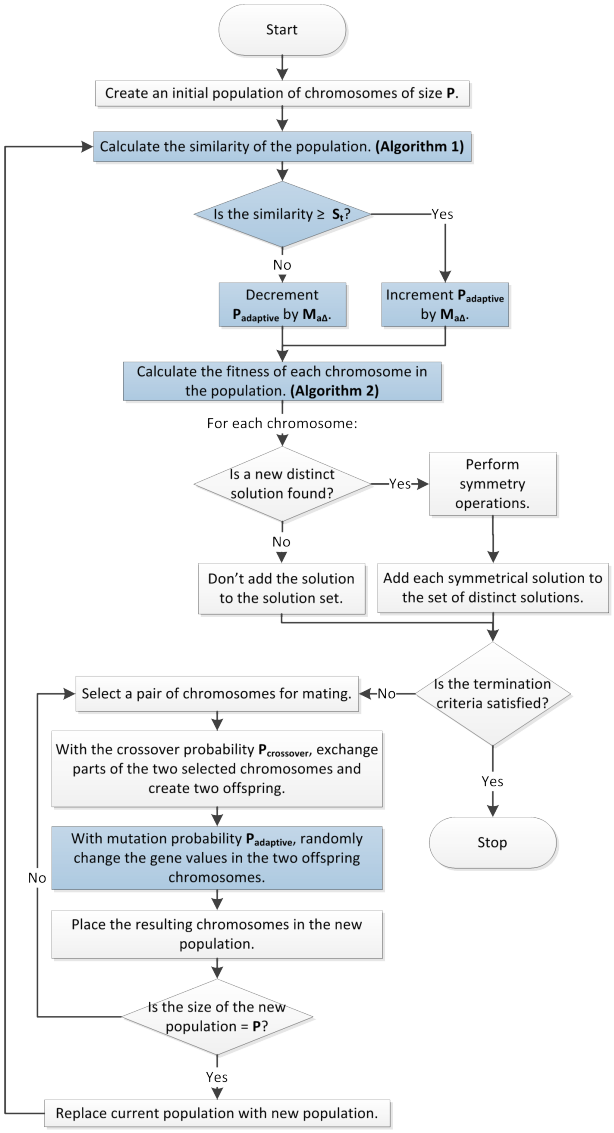

Fig. 3. Self-Adaptive GA Flowchart

TABLE II
Experimental Control Parameters

Variable	Value
Similarity Threshold (S_t)	15%
Adaptive Mutation Bounds (M_a)	$[1, 99]$
Adaptive Mutation Increment ($M_{a\Delta}$)	1
Crossover Probability ($P_{crossover}$)	70%
Cloning Probability ($P_{cloning}$)	30%
Adaptive Mutation Probability ($P_{adaptive}$)	$variable$
Population Size (P)	64

B. Self-Adaptive Mutation

Chromosome similarity is used as a main feature in our adaptation approach. The probability for the mutation operator is dynamically adjusted, while recombination ($P_{crossover}$) remains at a single static value as in traditional GA. Our experimental settings for the genetic operators are given in Table II.

In order to apply our self-adaptive mutation approach, a two step process is required. First, each generation's chromosomes' similarity must be evaluated. The similarity algorithm is detailed in Algorithm 1. Next, the mutation probability is adjusted accordingly in order to adapt. This allows the population to be diversified when it is required to improve the search.

When the chromosome similarity is less than the specified threshold (S_t), the population is too dissimilar (high diversity). This means the adaptive mutation probability ($P_{adaptive}$) must decrease, because the population is not inbred. For the next generation, the mutation probability is subtracted by $M_{a\Delta}$ indicated in Table II. After application of genetic operations and selection on the population, the next generation is populated and their chromosome similarity is calculated. If the chromosome similarity has increased, and if it is ever above the specified threshold, the population is inbred. $P_{adaptive}$ must increase by the fixed delta in the next generation to adapt.

Variations in chromosome similarity may result independently of the applied uniform mutation operator. Since the recombination probability ($P_{crossover}$) is 70%, the recombinant permutations are likely to be approximately 30% similar (due to cloning) unless uniform mutation has occurred on the cloned genes. Our experimental observations indicate that when using adaptive mutation the chromosome similarity will approach an equilibrium percentage approximately equal to the specified similarity threshold (S_t) as generations evolve.

C. Chromosome Similarity Algorithm

Chromosome similarity is calculated using a deterministic algorithm, presented in Algorithm 1. First, the chromosomes' genes are decoded and re-encoded into integer values and concatenated into one large integer. Next, the resultant array of integer representations is sorted. Our implementation uses the quick-sort algorithm provided by Java for minimal computational overhead. This gives the similarity algorithm its characteristic asymptotic behavior, as the complexity of the sorting function itself is of higher complexity than the rest of the main similarity calculation algorithm. Quick-sort runs in $O(n \log n)$ complexity, although it should be noted that in the theoretical worst case, quick-sort works in $O(n^2)$ which is highly unlikely. The main algorithm runs in $O(n)$ linear complexity independent of the sorting. Similar chromosomes have the same genes. A population with 100% similarity is completely identical. If there is a population such that $P = 4$, and two separate pairs of 2 chromosomes in that population are the same, the population also has 100% similarity. The emphasis with the similarity calculation is to isolate and distinguish the proportion of individually dissimilar chromosomes from the rest of the population.

```
Function Similarity(chromosomes)
    input : An array of chromosomes
    output: Fraction of chromosomes that are similar
    similar ← 0
    matched ← false
    length ← length of chromosomes
    // Use arbitrary sorting algorithm
    sorted ← Sort(chromosomes)
    for i ← 0 to length − 1 do
        if sorted[i] == sorted[i + 1] then
            similar ← similar + 1
            matched ← true
        else if matched then
            similar ← similar + 1
            matched ← false
        end
        // Check: last item is a match
        if matched and (i + 1 == length − 1) then
            similar ← similar + 1
        end
    end
    return similar/ length
end
```

Algorithm 1: Chromosome Similarity Evaluation Function

D. Fitness Function

Candidate solution fitness is evaluated by determining the number of collisions (C) between queens on the chessboard. This means that in a board state where one queen can attack another, two collisions result. When three queens can attack another, 6 collisions result. The algorithm for determining the fitness of a given board state is given in Algorithm 2, showing the mechanisms used to find collisions across diagonals and columns on the board. The overall board state fitness is calculated as $fitness = \frac{1}{C}$, where $fitness = 1$ if $C = 0$. Maximum fitness is achieved when $C = 0$ as there are no collisions, resulting in a distinct board solution. For example, in the classical 8-Queen problem, the ideal fitness is 1, and the worst case fitness is $1/56$.

E. Identifying Distinct Solutions

In each generation there is always a strong possibility that some of the chromosomes in the current population are valid solutions - any of the current generation's chromosomes with a fitness of 1 is some sort of solution. Whether it is distinct or not is unknown at first. When a solution is found, it must be compared to the list of previously found solutions to determine whether it is distinct. Symmetry operations are applied to each solution found to generate more distinct solutions based on the symmetrical nature of the chessboard. This

```
Function Fitness(chromosome)
    input : A single chromosome
    output: A fitness value for the chromosome
    collisions ← 0
    length ← length of the chromosome
    for i ← 0 to length − 1 do
        // Compare each gene
        j ← (i + 1) mod length
        while j != i do
            y_i ← chromosome[i]
            y_j ← chromosome[j]
            // Check: column queens
            if y_i == y_j then
                collisions ← collisions + 1
            end
            // Check: diagonal queens
            if abs((i − j) / (y_i − y_j)) == 1 then
                collisions ← collisions + 1
            end
            j ← (j + 1) mod length
        end
    end
    if collisions == 0 then
        return 1
    else
        return 1 / collisions
    end
end
```

Algorithm 2: Fitness Function

may further increase the likelihood that the new population's solutions have already been found previously. Therefore, indistinct solutions are discarded without having symmetry operations applied as their reconfiguration would already be identical to symmetric configurations of previously found distinct solutions. After saving newly found solutions, the new generation replaces the previous generation and the GA continues.

Symmetry operations consist of three $90°$ rotations, a horizontal reflection, and then 3 further $90°$ rotations of the reflected board. Each symmetrical board configuration is a candidate solution that is checked against the existing distinct solutions. It is discarded if it has already been found previously, because it is indistinct. Each distinct solution can potentially yield up to 7 additional distinct solutions depending on the placement of queens, giving 8 solutions in total.

V. Experiments

We measured the performance of our self-adaptive approach using an empirical study. Our methodology was to solve N-Queens for various values of N with a limited budget of generations. Number of function calls (*NFC*) is given as $P \times Generations$ where $P = 64$ as specified in Table II, as the fitness evaluation function is called P times per generation. Each experiment was repeated a multitude of times to ensure accuracy and collect aggregate statistics. Table III shows the number of generations permitted and the number of trials run for each value of N. Special budgets and trial multiplicities were set for $N = 32$ in order to accommodate the exceptionally large problem size.

TABLE III
Experimental Trials

N	Budget (Generations)	Trials
8	10,000,000	30
9	10,000,000	30
10	10,000,000	30
11	10,000,000	30
12	10,000,000	30
13	10,000,000	30
14	10,000,000	30
15	10,000,000	30
16	10,000,000	30
18	10,000,000	15
20	10,000,000	15
22	10,000,000	15
24	10,000,000	15
26	10,000,000	15
32	50,000,000	10

Our control used traditional GA with fixed mutation probabilities (M_c) of M_c = 1%, 5%, 10% to 100% (with a step size of 5%) to solve each N-Queens problem. These trials were run with the same multiplicity and budget as our self-adaptive approach. The goal of the control trials was to approximately determine the optimal M_c for finding the most solutions for each problem in the given time budget, and finding the first solution fastest. Our aim was to test our new approach against a reasonable approximation of the best possible performance that the traditional GA approach can provide. Optimized control parameters for mutation would be determined and used as a benchmark, similar to Hesser and Männer's study [15].

Given the increasingly large problem sizes, the number of tests were reduced in order to allow experimentation to complete within reasonable time frames. Descriptive statistics were gathered every 1000 generations, with mean calculated using the grand mean.

VI. Results and Analysis

The performance difference with respect to the number of found solutions can be seen in Table IV. The traditional GA approach performs marginally better in the range of small problems, which are most efficiently solved with deterministic methods. This range consists of values of $N \leq 15$. Our self-adaptive approach yields more solutions for values of N where $N > 15$. As N increases, the difference widens; the self-adaptive approach provides increasingly better results for this objective as the problem size increases.

We observed that there is a proportional inverse relationship between N and the best experimentally determined M_c. As N increases, the best M_c which yielded the most solutions decreases. The point this raises is that there is an

unfair advantage for the traditional GA over the self-adaptive approach, since the optimal static M_c changes as N changes. But we know that the ideal M_c for all problems are unknown. Generally, foreknowledge of the ideal M_c is not available in order to exploit this advantage [15], but in this case it was experimentally determined. Optimally tuned control parameters are critical and provide a major advantage. Still, the self-adaptive approach consistently outperformed traditional GA with optimized parameters for the large problem sizes, even as the parameters were adjusted to better suit the problem. Our conclusion is that the self-adaptive approach will continue to outperform the traditional GA method as N grows, based on the increasing gap in performance observed. The main reason is that even the most optimal value of the mutation rate remains unchanged during the search process. This is why traditional GA cannot compete with our self-adaptive scheme.

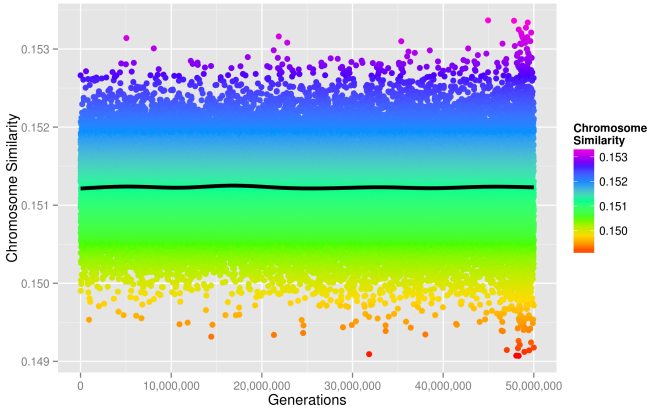

Fig. 4. Chromosome Similarity vs. Generations for the 32-Queen Problem (Self-Adaptive)

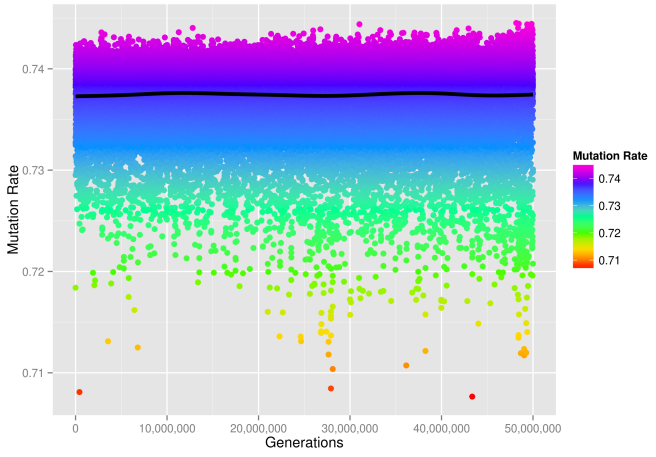

Fig. 5. Mutation Probability vs. Generations for the 32-Queen Problem (Self-Adaptive)

Our analysis of the algorithm shows that the self-adaptive approach adjusts the mutation probability throughout the duration of the solution search. In contrast to historical approaches, the adaptive approach does not adapt based on one characteristic of the problem, the stage of the search, position of the individuals on the landscape, or fitness values. Our results also show that adaptation occurs every generation, since stochastic factors routinely influence the similarity beyond or below the similarity threshold (S_t). This behavior can be seen in Figure 4, which shows that the chromosome similarity changes as a result of the mutation operator to approach the specified S_t. Figure 5 shows the corresponding changes in mutation probability over generations based on similarity. Comparatively, Figure 6 shows the similarity over generations of the most optimal traditional mutation operator found for the 32-Queen problem, $M_c = 0.65$. At this mutation probability, traditional GA tends toward a similarity of $25\% \pm 1\%$, with lower and upper extrema at 1.8% and 26.4%. This lies in stark contrast to the self-adaptive approach which tended toward the specified S_t (15%), using adaptive mutation probabilities that tended toward 73.7% with lower and upper extrema at 70.8% and 74.5%.

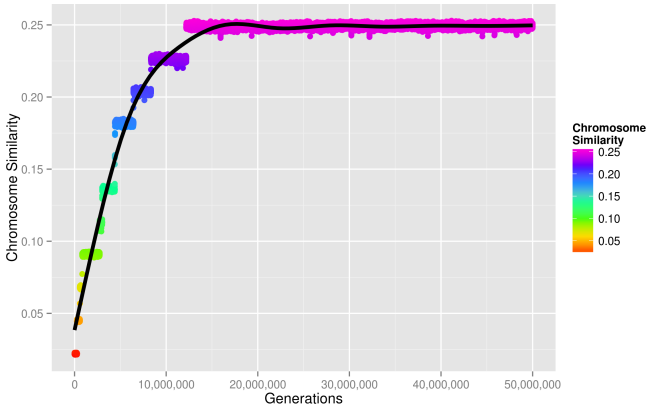

Fig. 6. Chromosome Similarity vs. Generations for the 32-Queen Problem (Traditional GA, $M_c = 0.65$)

The self-adaptive approach increases diversity and generates more diverse candidate solutions for efficient traversal of the areas of the problem landscape that contain undiscovered solutions. The diversity is increased because the search is prevented from regenerating identical chromosomes, flooding the population with clones. Controlling similarity allows the self-adaptive approach to strike a balance between convergence, exploration, and exploitation. Therefore, there is a reverse relationship between diversity and chromosome similarity which enables the diversity of the population to be controlled by the specified threshold (S_t). In traditional GA, when the chromosome similarity is too high, diversity is low and few solutions are found. This is evident by the poor performance yielded by traditional GA in our benchmarks where M_c is very small. There is no mechanism other than the constant background mutation that increases diversity among the population, since recombination of identical chromosomes results in a clone regardless. Conversely, when mutation is too frequent (such as in high values of M_c like 100%), similarity becomes low and the diversity of the population becomes too high to allow optima to be

TABLE IV
MOST N-QUEENS DISTINCT SOLUTIONS

N	Best M_c	Distinct Solutions Found		% Difference
		Best M_c	Self-Adaptive	
8	0.95	92	92	$\pm$ 0%
9	0.95	352	352	$\pm$ 0%
10	0.9	724	724	$\pm$ 0%
11	0.85	2,680	2,680	$\pm$ 0%
12	0.85	13,690	11,986	-12.45%
13	0.8	32,128	26,308	-18.12%
14	0.8	41,520	29,520	-28.90%
15	0.8	30,356	30,324	-0.11%
16	0.8	15,016	30,132	$+100.67\%$
18	0.75	16,392	27,120	$+65.45\%$
20	0.75	7,872	25,608	$+225.30\%$
22	0.7	6,008	25,376	$+322.37\%$
24	0.7	6,440	24,560	$+281.37\%$
26	0.7	6,280	23,008	$+266.37\%$
32	0.65	15,216	104,080	$+584.02\%$

TABLE V
FIRST N-QUEENS DISTINCT SOLUTION

N	Fastest M_c	Number of Generations		% Difference
		Fastest M_c	Self-Adaptive	
8	0.75	45	91	-50.55%
9	0.8	121	186	-34.95%
10	0.8	261	417	-37.41%
11	0.7	444	364	$+21.98\%$
12	0.65	457	463	-1.30%
13	0.7	448	582	-23.02%
14	0.65	494	609	-18.88%
15	0.65	598	513	$+16.57\%$
16	0.65	662	606	$+9.24\%$
18	0.65	911	688	$+32.41\%$
20	0.55	889	862	$+3.13\%$
22	0.5	1,234	1,211	$+1.90\%$
24	0.4	1,209	1,182	$+2.28\%$
26	0.45	1,599	942	$+69.75\%$
32	0.5	2,298	1,995	$+15.19\%$

converged upon. The search algorithm turns into a random walk approach. When solutions are converged upon in the self-adaptive approach, diversity increases as the mutation rate adapts in order to expand the search to other unexplored regions of the landscape. This behavior is the desirable characteristic of the self-adaptive approach that makes it ideal for highly multi-modal problems with many optima like N-Queens. Controlled similarity prevents the GA from being stuck in a plateau of similar local optima, while also preventing the search from being wildly dissimilar and random, failing to converge on global optima.

A. Testing Against Multi-Perspective Challenges

While our primary goal was to compare which approach generated as many solutions as possible on a fixed budget, the secondary aim of the N-Queens problem is to generate a single solution as quickly as possible. This can theoretically be performed in polynomial time using deterministic methods [23], [24], and is shown to be most efficient in our results presented in Table VI. When the problem size is very large this is nowhere near as fast or efficient as stochastic approaches.

TABLE VI
FIRST N-QUEENS DISTINCT SOLUTION – DETERMINISTIC

N	Number of Function Calls (*NFC*)		% Difference
	Deterministic [22]	Self-Adaptive*	
8	876	5,824	-84.96%
9	333	11,904	-97.20%
10	975	26,688	-96.35%
11	517	23,296	-97.78%
12	3,066	29,632	-89.65%
13	1,365	37,248	-96.34%
14	26,495	38,976	-32.02%
15	20,280	32,832	-38.23%
16	160,712	38,784	$+314.38\%$
18	743,229	44,032	$+1,587.93\%$
20	3,992,510	55,168	$+7,137.00\%$
22	38,217,905	77,504	$+4.92 \times 10^4\%$
24	9,878,316	75,648	$+12.96 \times 10^4\%$
26	10,339,849	60,288	$+17.10 \times 10^4\%$
32	2,799,725,104	127,680	$+2.19 \times 10^6\%$

**NFC* is given as $P \times Generations$, where $P = 64$.

TABLE VII
FIRST 32-QUEENS DISTINCT SOLUTION – ALL MUTATION RATES

M_c	Number of Generations	% Difference*
0.01	168,977	$+8,370.03\%$
0.05	18,961	$+850.43\%$
0.1	14,286	$+616.09\%$
0.15	9,590	$+380.70\%$
0.2	6,628	$+232.23\%$
0.25	4,563	$+128.72\%$
0.3	3,480	$+74.44\%$
0.35	3,548	$+77.84\%$
0.4	3,290	$+64.91\%$
0.45	2,310	$+15.79\%$
0.5	2,298	$+15.19\%$
0.55	9,143	$+358.30\%$
0.6	63,428	$+3,079.35\%$
0.65	4,321,895	$+2.17 \times 10^5\%$
0.7	36,873,976	$+1.85 \times 10^6\%$
0.75	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
0.8	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
0.85	50,000,000 (No Sol.	$+2.51 \times 10^6\%$
0.9	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
0.95	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
0.95	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
1.0	50,000,000 (No Sol.)	$+2.51 \times 10^6\%$
Self-Adaptive	1,995	$Avg. = +8.92 \times 10^5\%$

**% Difference* is given as Generations $\frac{M_c - \text{Self-Adaptive}}{\text{Self-Adaptive}} \times 100\%$

Table VII shows how the self-adaptive approach compares to each individual M_c used to solve the 32-Queens problem with traditional GA. 32-Queens is the largest and most difficult problem of our experimental dataset. In this exemplar, the self-adaptive approach beats every mutation probability tested including the most optimal M_c. Compared across each fixed mutation rate, on average the self-adaptive approach

is 891917% better, which represents a staggering difference in performance. Compared to a 'sensible' value which is typically used for most problems without prior knowledge of the landscape (such as $M_c = 0.7$ [17]) the difference in performance is very prominent at 1848220%.

Table V shows our results for how many generations were required for the self-adaptive approach to find the first solution to each N-Queens problem, compared to the best possible M_c used in traditional GA. The experimental conditions were the same as the previous experiment, with data collected concurrently from the same trials. Our results show that the self-adaptive approach was able to generate a solution quicker than traditional GA, even given the same unfair advantage presented in the previous experiment. We show that even with the best possible values of M_c to produce the fastest single solution, the self-adaptive approach outperforms the traditional GA in the problems where $N > 14$. The conclusion is that even in terms of simultaneously finding the most solutions overall, and finding the first solution fastest, our self-adaptive approach prevails.

VII. Conclusions and Further Study Remarks

The main conclusion of our study is that for both benchmarking objectives (finding the most solutions, and finding the first solution fastest) for the N-Queens problem, our approach performs better than traditional GA even when optimized control parameters are used in its unfair benefit. This answers the question, why not use the traditional mutation operator (M_c)? Traditional GA with using a single M_c - even the most optimal one - cannot adapt to the stage of search that the algorithm is in. At some points in the traversal of the problem landscape, it is desirable to converge on a local or global optimum. In other circumstances, the search must be widened. When only using mutation as a fixed background operation naïvely, there is no choice. When we are using a self-adaptive approach, mutation probability can be changed and adapted based on the problem size, the characteristics of the problem landscape, problem difficulty, and time.

Exploring our self-adaptive mutation rate approach in other NP-Complete combinatorial and optimization problems such as the Traveling Salesman Problem or Constraint Satisfaction Problem may show that our approach is valid for a broad range of combinatorial problems with different characteristics. We conjecture that this is likely since other adaptive mutation methodologies have yielded performance improvements in various challenging problem domains.

Conducting a multivariate analysis and sensitivity analysis of tuning other parameters in conjunction with the variable adaptive mutation rate may yield further performance improvements. For example, variable population size based on the amount of inbreeding may yield even better results. This conjecture is based on the fact that with higher mutation rates, survival fitness decreases, resulting in population shrinkage [11]. Coupling the population size with the mutation rate and patterns of fitness in the GA population may yield a new avenue for future study.

References

[1] S. Tasan, S. Tunali, "A review of the current applications of genetic algorithms in assembly line balancing," it Journal of Intelligent Manufacturing, vol. 19, pp. 49-69, Springer US, 2008.

[2] P. Larraaga et al., "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," *Artificial Intelligence Review*, pp. 129-170, Kluwer Academic Publishers, 1999.

[3] K. Jong and W. Spears, "Using Genetic Algorithms to Solve NP-complete Problems," *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 124-132, George Mason University, USA, 1989.

[4] K. Crawford, "Solving the N-queens problem using genetic algorithms," *Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's*, SAC '92, pp. 1039-1047, ACM, New York, USA, 1992.

[5] A. Homaifar, J. Turner and S. Ali, "The N-queens problem and genetic algorithms," *Southeastcon '92, Proceedings., IEEE*, pp. 262-267, Birmingham, AL USA, 1992.

[6] P. Andrews, "An investigation into mutation operators for particle swarm optimization," *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 1044-1051, 2006.

[7] A. Tuson and P. Ross, "Adapting operator settings in genetic algorithms," *Evolutionary computation*, vol. 6, no. 2, pp. 161-184, The MIT Press, 1998.

[8] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67-82, IEEE, 1997.

[9] M. Srinivas and L. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, no. 4, pp. 656-667, IEEE, 1994.

[10] M. Srinivas and L. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6 pp. 17–26, IEEE, 1994.

[11] D. Goldberg and J. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95-99, Springer, 1988.

[12] D. Goldberg, "Genetic and evolutionary algorithms come of age," *Communications of the ACM*, vol. 37, no. 3, pp. 113-119, ACM, 1994.

[13] K. Jong, "Adaptive system design: a genetic approach," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 10, no. 10, pp. 566-574, IEEE, 1980.

[14] Z. Ye, Z. Li and M. Xie, "Some improvements on adaptive genetic algorithms for reliability-related applications," *Reliability Engineering & System Safety*, vol. 95, no. 2, pp. 120-126, Elsevier, 2010.

[15] J. Hesser and R. Männer, "Towards an optimal mutation probability for genetic algorithms," *Parallel Problem Solving from Nature*, vol. 496, pp. 23-32, Springer Berlin Heidelberg, 1991.

[16] J. Coyne and R. Paton, "Genetic algorithms and directed adaptation," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 5, pp. 533-541, 1986.

[17] M. Negnevitsky, *Artificial Intelligence: A Guide To Intelligent Systems*, Pearson Education Limited, edition 2, 2005.

[18] R. Clark, J. Stainer, H. Haynes, R. Buckner, JE. Mosier, AJ. Quinn and others, *Medical and genetic aspects of purebred dogs*, Veterinary Medicine Publishing Co., 690 South 4th Street, 1983.

[19] P. Mannucci and E. Tuddenham, "The hemophiliasfrom royal genes to gene therapy," *New England Journal of Medicine*, vol. 344, no. 23, pp. 1773-1779, Mass Medical Soc., 2001.

[20] D. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Urbana*, vol. 51, pp. 61801-2996, 1991.

[21] Masehian, Ellips et al., "Landscape analysis and efficient metaheuristics for solving the n-queens problem," *Computational Optimization and Applications*, vol. 56, pp. 735-764, 1991.

[22] C.S. Pearson and M.S. Pearson, "A140450 The count of how many queens must be placed tentatively onto a board while seeking a first solution to the "N-Queens on an N x N chessboard" puzzle," *The Online Encyclopedia of Integer Sequences*, Aug. 2008. [Online]. Available: http://oeis.org/A140450. [Accessed Jan. 10, 2014].

[23] B. Bernhardsson, "Explicit solutions to the N-queens problem for all N," *ACM SIGART Bulletin*, vol. 2, pp. 7, 1991.

[24] R. Sosic, J. Gu, "A polynomial time algorithm for the N-Queens problem," *ACM SIGART Bulletin*, vol. 1, pp. 7-11, 1990.

[25] R.G. Spallek, T.B. Preußer and B. Nägel, "QUEENS@TUD: The World Record by FPGAs!," *Technische Universität Dresden*, Aug. 2009. [Online]. Available: http://queens.inf.tu-dresden.de. [Accessed Jan. 13, 2014].