On Optimal Population Size of Genetic Algorithms

Jarmo T. Alander
Laboratory of Information Processing Science
Department of Computer Science
Helsinki University of Technology
Otakaari 1, SF-02150 Espoo, Finland
E-mail: ja@cs.hut.fi

Abstract

In this paper we describe the result of our experiments to find the optimum population size for genetic algorithms as function of problem complexity. It seems that for moderate problem complexity the optimal population size for problems coded as bitstrings is approximately the length of the string in bits for sequential machines. This result is also consistent with earlier experimentations. In parallel architectures the optimal population size is larger than in the corresponding sequential cases but the exact figures seem to be sensitive to implementation details.

Keywords: *genetic algorithms, optimization, operations research, population size, programming*

1 Introduction

The most attractive features of genetic algorithms (GA) are their simplicity and robustness. In spite of the seemingly simple processing, the GAs are known to be good in solving some problems that are known to be hard. The simplicity, flexibility, parallelism, and the good problem solving capability were the most important facts that made us to experiment with GAs in order to finally find methods to solve some robot control problems.

The implementation of genetic algorithms includes determination and optimization of a number of details, procedures and parameters. One of the most important parameters is the population size, which is the topic of this paper.

The idea behind the work has been to find methods that can solve difficult robot control problems such as adaptation, classification, and task planning in a more or less distributed computer environment. The work is part of our M^3-project (Modular, Multiarm, and Multipurpose robot cell) [Ala90, Ala91].

1.1 Related work

The fundamentals of genetic algorithms can be found in David Goldbergs book [Gol89a]. A number of conferences devouted to the theory and applications of genetic algorithms have also been held [Sch89, BB91, SM91]. In addition, Lawrence Davis gives a good overview of GA applications [Dav91]. He also suggests a practical procedure to attack new problems by combining genetic algorithms with existing methods into, as he calls, hybrid GA: The pure basic genetic algorithm is usually not the best possible algorithm to solve any specific problem if a good specific algorithm already exists. There is, however, a good chance to create a better algorithm by combining the best features of existing conventional and genetic algorithms. Especially the adaptive search feature of GA seems beneficial.

John J. Grefenstette has used genetic algorithms to find the optimal parameters of genetic algorithms [Gre86]. He has tried to find the optimal values of six parameters: population size (n), crossover rate (c), mutation rate (m), renewal rate (r), scaling window (w) [generations], and elitist/pure selection (s). The total number of parameter combinations, i.e. the number of different possible genetic algorithms was 2^{18}. The "standard" genetic algorithm based on De Jong's work [Jon75] has parameter values: $GA_s(n, c, m, r, w, s)$ = GA(50, 0.6, 0.001, 1.0, no scaling, elitist). After Grefenstette the best genetic algorithm with respect to his test function environment is GA_{JJG} = GA(30, 0.95, 0.01, 1.0, 1, elitist).

Goldberg have analyzed theoretically the optimal population sizes [Gol89b]. After his analysis the optimal population size increases exponentially and is rather large for even moderate chromosome lengths. For practical applications these results seem rather pessimistic ones and as Goldberg states: "Too few empirical studies have been performed to know whether the theory provides quantitatively accurate predictions." [Gol89b]

Our genetic algorithms somewhat differ from that of the previously mentioned work. That is why our first step towards the application of genetic algorithms is to try to find out what are the optimal values of the different parameters, before applying genetic algorithms to some elementary problems of robot control. In our previous work we have also used GA to find some optimal parameters of GA [Ala91]. The values of the parameters are shown in table 1.

The surprisingly rapid convergence of the parameters to be optimized made us to make more analyses and experiments. In this paper we describe the work on the population size, which is for execution efficiency the most interesting parameter of GA when running GA on a sequential machine.

0-8186-2760-3/92 $03.00 © 1992 IEEE

parameter	*value*
crossing rate	maximum (uniform)
mobility	maximum (15)
mutation frequency	0-2 specimens/generation
population size	≈ 40
subpopulations	3-4
elitism	$\approx 1/4$ best always taken
mutation type	3 (all ones)
min. breeding distance	8

Table 1: Optimal parameters of our genetic algorithm [Ala91].

2 The genetic algorithm

Our genetic algorithm package is written in C++[Str85] and it currently contains about 4000 lines of code. The most important data structures and procedures of our genetic algorithm are as follows. The individual solution candidates are represented by objects of `specimen`-class. Specimens are arranged in an array by descending order of fitness. The array is an attribute of `population`-class. The specimen array is processed starting from the best specimens: Somewhat simplified the procedure in C++ is the following:

```
for (int i=0; i<n-1; i++) {
    int m = mobility(i,n);
    save(S[i]);
    breed(S[i],S[m],Boys);
    save(Boys[0]);
    save(Boys[1]);
}
takeBest(S,n);              // update array S
```

where `S` is the specimen array, `n` is the number of specimens, `mobility()` gives the (random) index of the other parent, `breed()` breeds the specimens producing two offspring (`Boys`), `save()` saves specimen in fitness order for `takeBest()`, which takes back the `n` best specimens into the array `S`.

2.1 Parameters

Uniform crossover means that every field has equal probability (0.5) to be selected from either parent [Sys89].

Mobility gives the range of random distance between parents:

```
mobility(i,n) = min(n, i + random(m));
```

where `m` is the mobility parameter of GA.

Mutations are handled by breeding `mfreq` specimens with special mutation chromosomes. There are several possible types of mutation chromosomes:

- alternative all zeros and all ones chromosomes
- pseudorandom chromosomes
- random chromosomes having reverse bit probabilities i.e. the probability of the given bit is the reverse of the probability of the the given bit position of the total population

In our experimentations we have used the all ones mutation chromosomes.

Elitism gives the relative amount of (sorted) best items of `S`. The sorting of `s` takes time but it also makes it possible to favour arbitrarily small fitness differences among the most best specimens. An alternative solution to this selection problem has been to make fitness value windows and scalings according.

3 Population size

There are several ways of estimating and choosing a reasonable population size. The most commonly used method according to GA literature has been to set population size equal to the usually well working value 50. In the following sections we will look some other ways.

3.1 Linear algebra

From linear algebra we know that in order to cover each point of given n dimensional space, we need at least n base vectors $\vec{b_i}$. The number is exactly n when the set is linearly independent and every vector $\vec{r}$ of the given n dimensional space can be expressed as a linear combination of the n base vectors:

$$\vec{r} = \Sigma_{i=1}^{n} c_i \vec{b_i}$$

On the one hand the crossover operator generates new combinations and that decreases the need of separate "base" chromosomes. On the other hand it also takes time to reach the proper combinations and in deceptive cases the progress done by the crossover operator may be more or less random. The mutation operator also reduces population size by introducing possible new base vector candidates, but it may take arbitrarily long time before we reach the absent or extinct field values or alleles and still longer to compose a good solution combination of values. Usually the mutation probability is considerably low so that the time needed is relatively long. From the above reasoning it should be obvious that the optimal population size $S_{opt}(n)$ for small values of n may be approximately equal to n:

$$S_{opt}(n) \approx n \, .$$

3.2 Schema processing

After Goldberg [Gol89b] the number of schemata[1] processed effectively is proportional to n^3, where n is the population size. This seems to justify the selection of ever larger population sizes. Goldberg [Gol89b] has, however, shown that under certain assumptions there exists (considerably large) optimal values of n.

[1] By schema a subset of gene values is denoted.

3.3 Population size in nature

It is known from nature that large populations are more stable and resist evolution more than small populations perhaps founded by only a few colonists. It is often so that these small isolated populations that are peripheral to the main range of a species are highly divergent. This so called founder effect has also been confirmed by laboratory tests [Fut86][p. 238].

3.3.1 Polyploidy

The role of diploidy and polyploidy in general seems to be to stabilize and conserve genetic information. Hardy-Weinberg theorem gives the asymptotic frequences of different alleles and proofs in a simple way that even the dominant alleles are not able to totally replace recessive alleles in statistical sense. Most of the genetic algorithms lack polyploidy, which could be a means to avoid premature convergence especially in dynamic and oscillating situations. Goldberg [Gol89a] describes some work on polyploidy in GA and analyses the influence of polyploidy to schemata changes.

4 Experimentations

The basic parameters of our experimentations are the optimal ones from our earlier work [Ala91] shown in table 1. Other parameters are shown in table 2.

parameter	*value*
mutation frequency	2 per generation
subpopulations	4
elitism	1/4 best always taken
min. breeding distance	1

Table 2: The fixed parameters of the tested genetic algorithms.

The population size distribution is tested by the following method: Every population size ranging from zero to 127 specimens were used to find a bit pattern. The test was repeated 100 times for each population size using different random sequences. The testing took several days and nights of processor time on our SPARCstation[2] computer.

4.1 The test problem

The test problem used was to find a given subbitstring of the following 32 bit long test pattern P:

$$P = 00001111111000000111110000111001$$

The subbitstring was given by a mask vector m. In practise the value m was chosen to take the n rightmost bits of the test string. The fitness function f was the Hamming distance H, i.e. the number of differing bits, between P and the specimen chromosome C:

$$f(C) = H(P, m \odot C),$$

where $\odot$ is the bitwise and-operator for masking.

By problem complexity we mean the length of the subbitstring, n.

4.2 Execution time model

The observed execution time t versus population size s and problem complexity c are shown in fig. 1. In order to find an approximation formula to the optimum population size versus problem complexity each curve was approximated by a least squares line $t(c)$. The slopes of these lines $k_1(c)$ are show in fig. 2.

$$t(c) = k_0(c) + k_1(c)s$$

In fig. 3 is shown the overhead term $k_0(c)$. A third order polynomial was used to approximate $k_0(c)$:

$$k_0(c) = a_0 + a_1c + a_2c^2 + a_3c^3$$

In the same way a straight line was used to approximate $k_1(c)$:

$$k_1(c) = b_0 + b_1c$$

The model of execution time t is now

$$t = t(c, s) = k_0(c) + k_1(c)s$$

For small population sizes it seems that the optimum population size s_{opt} is approximately equal to the complexity of the problem c, from this assumption it follows that

$$t_{\text{opt}} = k_0(c)+k_1(c)c = a_0+(a_1+b_0)c+(a_2+b_1)c^2+a_3c^3.$$

The t_{opt} curve is shown in the fig. 4.

In figure 5 you can see the minimum population size s_{min}. For population sizes less than the minimum value the convergence is poor and more or less chaotic.

4.3 Parallel execution model

GAs are more or less parallel in nature. We have not made test runs on any parallel architecture. Instead we have assumed, that the execution time t_∞ for (moderately) unlimited number of processors is:

$$t_\infty(n) = t_1(n)/\sqrt{n}$$

where $t_1(n)$ is the time needed in our one processor test and n is the size of the population. The optimal value of population size is not very distinct, but the value $\approx 2n$ seemed reasonable in most cases.

4.4 Discussion

From the above experimentations we can estimate, that the optimal population size S_{opt} was approximately somewhere in the range:

$$^2log(N) \leq S_{\text{opt}}(N) \leq 2\ ^2log(N)\ ,$$

where N is the (small) number of all (valid) gene combinations. In general

$$N = \prod_{i=1}^{n} n_i$$

[2] SPARCstation is a trademark of Sun Microsystems, Inc.

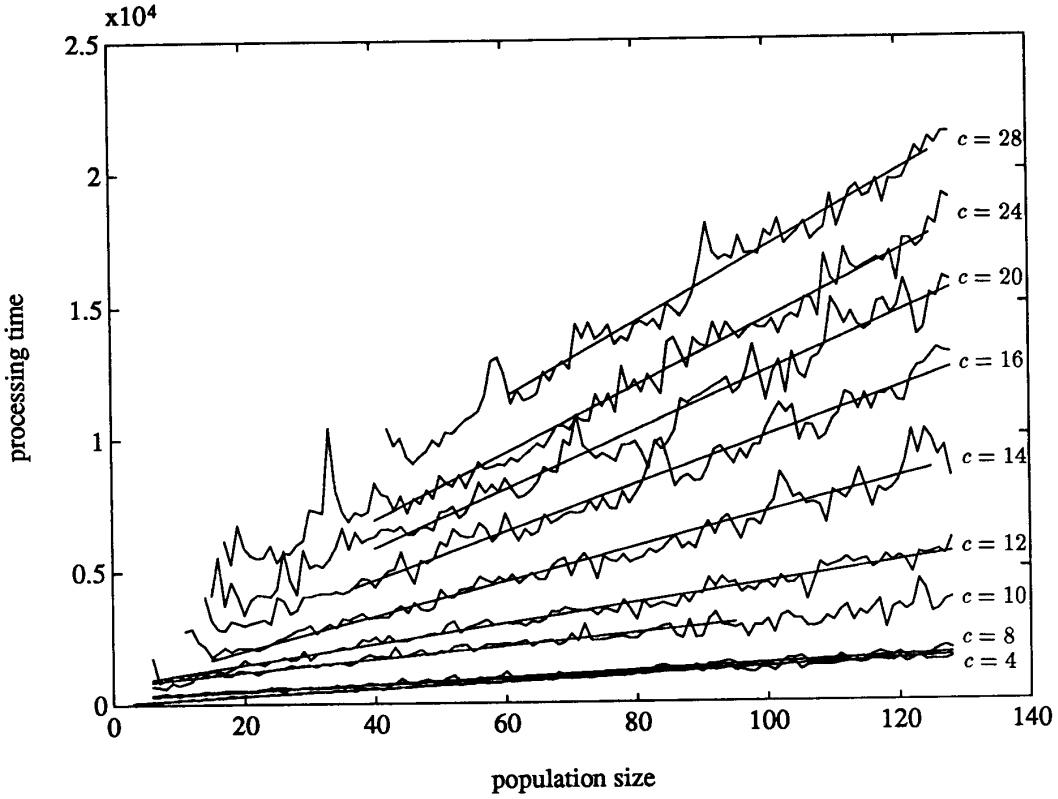

Figure 1: Execution time t versus population size s and problem complexity c (time in relative units).

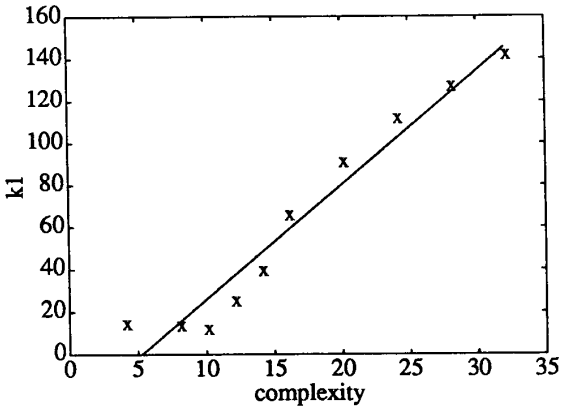

Figure 2: The slope of the execution time curve k_1 vs. problem complexity c.

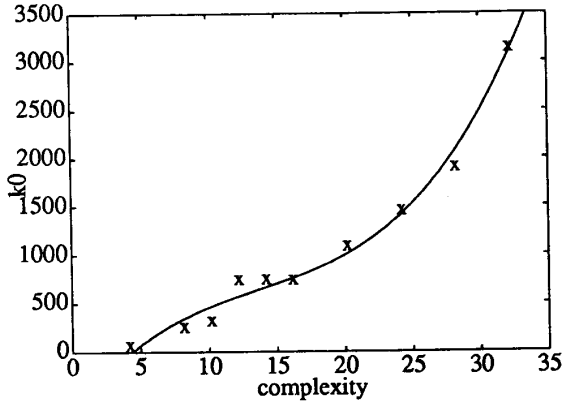

Figure 3: The overhead execution time k_0 vs. problem complexity c.

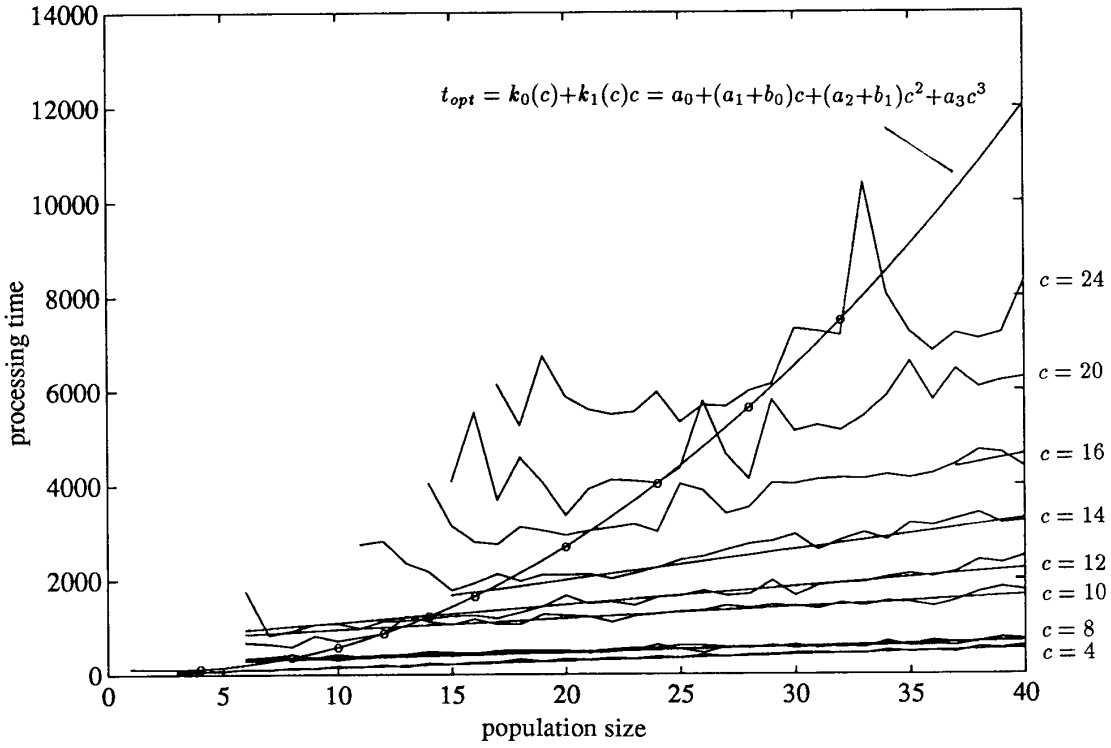

Figure 4: Predicted optimal execution time t_{opt} vs. problem complexity c. Measured times are also shown.

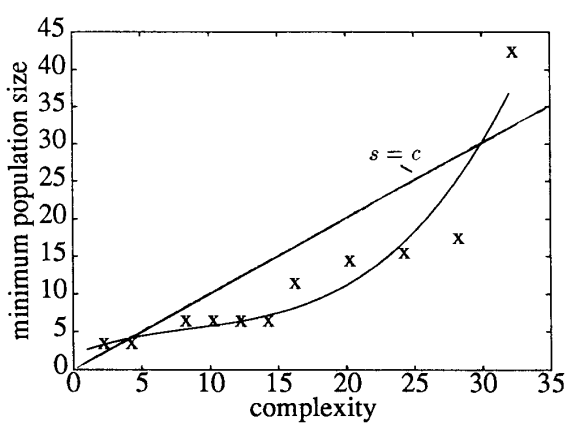

Figure 5: The minimum population size S_{min} vs. problem complexity c.

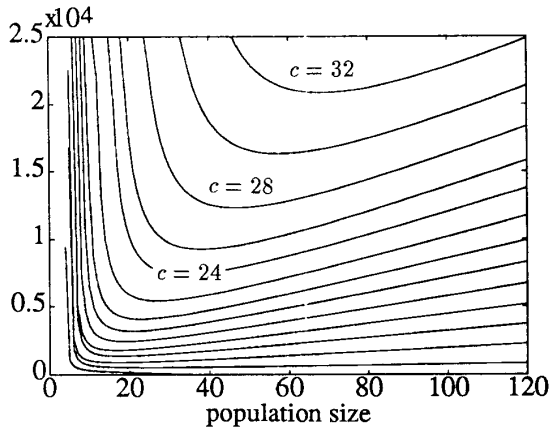

Figure 6: An ideal execution time t curve vs. population size s and complexity c (time in relative units).

where n_i is the number of valid values of gene position i and n is the number of genes. In our test case $N = 2^n$ in which case the above approximative formula is:

$$n \leq S_{opt}(n) \leq 2n \ .$$

Goldberg's mathematical analysis [Gol89b] gives a simple advice to parallel implementation of genetic algorithms: "choose as large a population size as you can". This is obvious, but the result is very sensitive on the model used. If we replace our square root model by a more optimistic architecture model, then the optimum population size rapidly grows.

5 Conclusions and Future

In this paper we have described a set of experimentations made in order to find the optimal population size for genetic algorithm for problems of varying complexity. It seems that for small population size n $S_{opt}(n)$ is:

$$n \leq S_{opt}(n) \leq 2n \ .$$

The testing was done on a sequential machine. The genetic algorithms, however, are parallel to a great extent so that optimal population size may well be greater than that used in our tests. After Goldberg the population size increases exponentially and is very large especially for parallel architectures [Gol89b]. The founder effect, however, may make very large population sizes less appealing if fast convergence and great divergence is aimed at.

Our experimentations were done using only one test problem. A more thorough analysis should cover a wide range of different problem types. The complexity of the test problem was also moderate and further studies should cover more complex problems.

Acknowledgements

The financial support of the Finnish Technological Development Center (Tekes) is gratefully acknowledged. Thanks also to Mr. Jouko Seppänen for his kind help with the proofreading of the manuscript.

References

[Ala90] Jarmo T. Alander. A modular, multipurpose, and multiarm assembly robot cell design. In Horváth Imre and Lehotzky József, editors, *Intertechno '90 Conference*, pages 10 – 19, Budapest, 11. - 14. Sept. 1990. Gépipari Tudományos Egyesület.

[Ala91] Jarmo T. Alander. On finding the optimal genetic algorithms for robot control problems. In *Proceedings IROS '91 IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, volume 3, pages 1313–1318, Osaka, 3.-5. Nov. 1991. IEEE Cat. No. 91TH0375-6.

[BB91] Richard K. Belew and Lashon B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, 13.-16. July 1991. Morgan Kaufmann Publishers.

[Dav91] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[Fut86] Douglas J. Futuyma. *Evolutionary Biology*. Sinauer Associates, Sunderland, Massachusetts, 2 edition, 1986.

[Gol89a] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.

[Gol89b] David E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In Schaffer [Sch89], pages 70–79.

[Gre86] John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–128, 1986.

[Jon75] K. A. De Jong. *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[Sch89] J. David Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*, Georg Mason University, 4.-7. June 1989. Morgan Kaufmann Publishers, Inc.

[SM91] H. P. Schwefel and R. Manner, editors. *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991. (Proceedings of the 1st Workshop on Parallel Problem-Solving from Nature (PPSN1), Dortmund, 1.-3. Oct. 1990).

[Str85] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, 1985.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In Schaffer [Sch89], pages 2–9.