Improving the Performance of Genetic Algorithms with a Gene Bank

Tapio Tyni and Jari Ylinen
Kone Elevators Research Center
P.O.BOX 677
05801 Hyvinkää, Finland
tapio.tyni@kone.com
jari.ylinen@kone.com

Abstract

Before the fitness of a chromosome is being evaluated with a – possibly computationally heavy - fitness function, it is first checked if the chromosome and the associated fitness value can be found from a Gene Bank. Only if the Gene Bank does not contain the chromosome to be evaluated the fitness function is activated. In that case the evaluated chromosome and its fitness value are stored into the Gene Bank. As the genetic search proceeds generation by generation the population becomes more and more homogenous concentrating around the solution of the problem; the probability to find a chromosome from the Gene Bank increases accordingly. Tests have proven that Gene Bank method improves the speed of the genetic search around 65% as the number of chromosomes actually needed to evaluate is reduced to one third when compared to the ordinary Genetic Algorithm. The implementation of the Gene Bank is based on hash tables and bi-directional circular linked lists to ensure that the accessing of the bank is swift. The speed of genetic search is especially important in real time applications where the time to achieve a solution is limited, as is the case in the Elevator Call Allocation problem, where we have successfully applied Genetic Algorithms enhanced with the Gene Bank technique.

Introduction

In many real life optimization problems the evaluation of the quality of a solution alternative is a tedious and computationally heavy task - the fitness function is often a simulation model of the target system. While Genetic Algorithms (GA's) are good to find high quality solutions to combinatorial problems they are considered as slow and inefficient in the usage of computational resources. Thus GA's are most often seen in offline applications instead of real world real time problems where the time available to achieve a solution is strictly limited.

In this paper we present a method to improve the performance of a binary coded Genetic Algorithm. The goal is to reduce the computational effort typical for real life problems by means that permit evaluation of each chromosome only once during the genetic search. The method - which we call Gene Bank - saves each evaluated chromosome and its fitness value to a store that can be accessed quickly and effectively when evaluating subsequent chromosomes. The operation of the Gene Bank is surveyed theoretically during epochs when population is in a steady state, i.e. during periods when population has converged to an extreme point or a basin on the problem search space. A computer experiment with a simple test problem is introduced and the theoretical results are compared with the experimental ones. The both results indicate that the adopted approach to model the behavior of the Gene Bank is appropriate. The Gene Bank technique is capable to reduce substantially the computational effort as well as search time in problems with computationally complex fitness functions.

Steady state behavior of the population and Epochal Evolution

While Genetic Algorithm proceeds towards the solution of the problem the population has typically long periods - epochs - when it wanders around a local extreme point until a successful mutation concentrates the search to a new region on the problem search space. When population lingers around an extreme it is said to be in a steady state [Dav91]. At the steady state the genes characteristic for the extreme have propagated over the population which becomes homogenous and dominated by the chromosome representing the local extreme. In this case crossing over is ineffective and the major way to get diversity to the population is mutation. In [vNC98] a concept of Epochal Evolutionary Search is presented. The approach states that epochal evolution is a system of “punctuated equilibrium” possessing a subbasin and portal architecture underlying the search space structure, figure 1. Population crawls and diffuses in a subbasin until a portal or tube to another subbasin is found (if goal is to minimize intuitively the tube is to a lower subbasin).

Figure 1 illustrates the case when the population spends time in subbasins $S_0 \in B^{<i>} \in S$. The subbasin consists of the neighborhood region around S_0 covered by the mutations performed at S_0. But what kind of mutations takes place and what is the subbasin covered by these mutations?

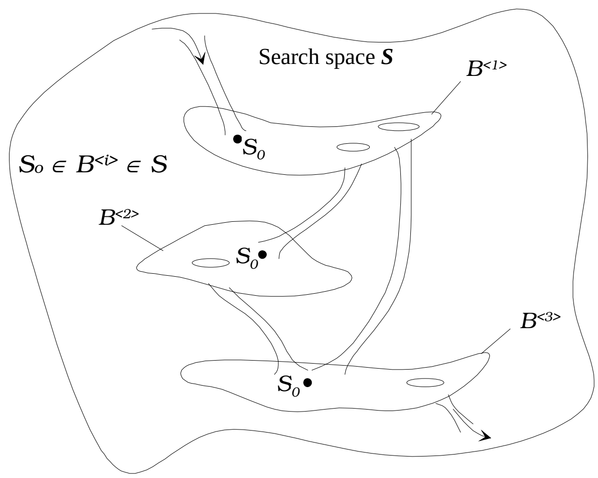


Figure 1. Subbasin – portal structure of the search space

Let G denotes the number of genes in the chromosome; M the number of genes that are mutated coincidentally in a chromosome and p_m the mutation probability, i.e. the probability that a gene is being mutated. With these parameters the *probability* or *proportion* that M-genes are mutated coincidentally in a chromosome comes from binomial distribution:

$$P^{<M>}(G,M,p_m) = \binom{G}{M} p_m^M (1-p_m)^{G-M} \qquad (1).$$

Equation (1) defines the probability or proportion of different *mutation orders M*. If all the genes in the chromosome have the same allele range r then the number of all possible mutations $N^{<M>}$ in a mutation order M is

$$N^{<M>}(G,M,r) = \binom{G}{M} r^M \qquad (2).$$

Here we define the *most probable neighborhood* close to $S_0 \in B$ as a region in subbasin B that is covered by the mutation orders M that are most likely to occur.

To give impression of practical figures table 1 summarizes some results calculated with parameters $G = 16$, $p_m = 0.02$ and $r = 26$. These parameters are used also in the example problem later in this paper. To interpret these figures, during an epoch in a genetic search, with the given parameters, 72.4% of the chromosomes created are not mutated at all (M=0). This fraction of chromosomes in the

M	$P^{<M>}(G,M,p_m)$	$N^{<M>}(G,M,r)$
0	0,724	1
1	0,236	416
2	0,036	81120
3	0,003	9842560
4	0,0002	831696320
5	0,00001	51897850368

Table 1. Example figures

population is identical and the re-evaluations are thus unnecessary waste of resources. Further 23.6% of chromosomes are predisposed to 1st order mutations (M=1) and there are totally 416 different ways to make such 1-gene mutation. Similarly 3.6% of the produced chromosomes have two genes mutated and there are 81120 possible ways to make 2nd order mutation. The *most probable neighborhood* in the given example might comprise mutation orders M=0,1,2,3. When M > 3 the mutation probability approaches rapidly zero.

In [Whi89] and [Dav91] a *Steady State without Duplicates* method is presented to avoid duplicate chromosomes and their evaluations. The created chromosome is accepted only if the chromosome does not already exist in the population. The method effectively increases the "quality" of the genetic operations since the same chromosome is not included twice - the neighborhood around the S_0 is explored more effectively. The drawback of the method is the numerous comparisons required. Every created chromosome has to be compared against each chromosome in the population before accepted; this may require several trials before an acceptable chromosome is produced.

The Gene Bank approach

Another approach to avoid the re-evaluation of already evaluated chromosomes is to save the chromosome and the associated fitness value (and possibly some other related information) into a pool – into a Gene Bank - from where the evaluated chromosomes can be accessed quickly and effectively later during the genetic search.

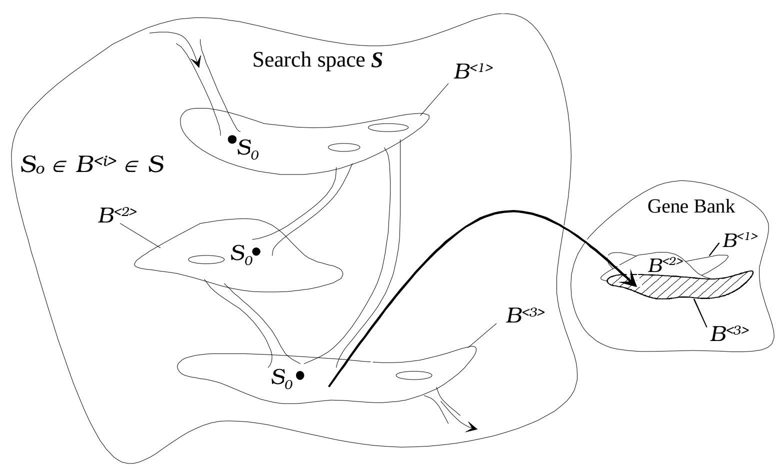


Figure 2. Sampling chromosomes into the Gene Bank

Before evaluating a chromosome it is first examined if the chromosome could be found from the Gene Bank. Only in the case the Gene Bank does not contain the chromosome it is evaluated and then inserted into the Gene Bank. Figure 2 illustrates the case when the search remains in subbasin $S_0 \in B^{<3>} \in S$. In the very beginning of the epoch the Gene Bank contains only the solution alternative $S_0 \in B^{<3>}$. While the mutation driven exploration proceeds in the subbasin $B^{<3>}$ the Gene Bank is gradually fulfilled with the most probable neighborhood of the S_0. The new chromosomes belonging to $B^{<3>}$ displace the chromosomes collected earlier during the search. This is illustrated with subbasins $B^{<1>}$ and $B^{<2>}$ in the Gene Bank. The older solutions are no more referenced since the search now takes place in the subbasin $B^{<3>}$. The displacement of older chromosomes is a desirable feature reducing the memory capacity requirements of the Gene Bank – most of the chromosomes created do not deserve survival in the Gene Bank through the entire search. In principle the Gene Bank should contain room for the whole search space S. However, recalling the most probable neighborhood, it is sufficient to allocate space only for the mutants that are most likely.

Let us derive the probability that a produced chromosome can be found from the Gene Bank during an epoch the search remains in a subbasin. The starting point is the number of M-genes coincidental mutations in a chromosome and the probability or relative proportion $P^{<M>}$ of such mutations. The probability that a chromosome with M-mutated genes can be found from the Gene Bank at generation k after beginning of a new epoch is

$$p_k^{<M>} = \frac{n_k^{<M>}}{N^{<M>}} \qquad (3),$$

where $n_k^{<M>}$ is the number of M-mutation order chromosomes in the Gene Bank at generation k. The number of chromosomes with M-mutated genes in the Gene Bank after generation k is

$$n_{k+1}^{<M>} = n_k^{<M>} + \Delta n_k^{<M>} \qquad (4),$$

where $\Delta n_k^{<M>}$ is the number of new chromosomes during generation k. If elitism is used and fraction of λ best chromosomes ($0 \le \lambda \le 1$) of the whole population N_P are selected to produce the next generation the expected number of generated M-mutation order chromosomes is

$$R^{<M>} = (1-\lambda) \cdot N_P \cdot P^{<M>} \qquad (5).$$

The number of new chromosomes $\Delta n_k^{<M>}$ in (4) is actually the number of chromosomes that *can not be found* from the Gene Bank during processing the generation k. The expected number of chromosomes that can not be obtained is

$$\Delta n_k^{<M>} = \bar{p}_k^{<M>} R^{<M>} = (1-p_k^{<M>}) \cdot R^{<M>} \qquad (6),$$

where $\bar{p}_k^{<M>}$ is the probability that a chromosome is *not found* from the Gene Bank. Substituting (6) to (4) and using (3) gives the number of chromosomes in the Gene Bank after k generations from start of a new epoch

$$n_{k+1}^{<M>} = n_k^{<M>} + (1-\frac{n_k^{<M>}}{N^{<M>}}) \cdot R^{<M>} \qquad , n_0^{<M>} = 0 \qquad (7).$$

The first order difference equation (7) is not in closed form and not suitable for analytical calculations. Solving and letting k_{S0} be the starting generation of a new epoch yields to closed form

$$n_{k-k_{S0}}^{<M>} = N^{<M>}(1-e^{-\frac{R^{<M>}}{N^{<M>}}(k-k_{S0})}) \qquad (8).$$

So far we have considered the behavior at the level of M-mutated genes per chromosome. To avoid the M-mutation limitation we have to recall (1), which gives the proportion of M-mutated genes in a chromosome. Using (8) and (3) and scaling each M-mutation probability with $P^{<M>}$ and summing up gives the probability to find any chromosome from the Gene Bank after a new epoch in the search has started

$$p_{k-k_{S0}} = \sum_{M=0}^{\infty} P^{<M>}(1-e^{-\frac{R^{<M>}}{N^{<M>}}(k-k_{S0})}) \qquad , k \ge k_{S0} \qquad \Leftrightarrow \qquad (9a)$$

$$p_{k-k_{S0}} = \begin{cases} 0, & k = k_{S0} \\ P^{<0>} + \sum_{M=1}^{\infty} P^{<M>}(1-e^{-\frac{R^{<M>}}{N^{<M>}}(k-k_{S0})}) & , k > k_{S0} \end{cases} \qquad (9b).$$

From (9b) we notice that the chromosomes with mutation order M=0 are immediately available from the Gene Bank once a new epoch has started. The probability of other mutation orders approaches asymptotically their proportion $P^{<M>}$. The ratio $N^{<M>}/R^{<M>}$ may be interpreted as the "time constant" representing the rate that the probability to find a chromosome from the Gene Bank increases. The question how many terms are needed in the sum, i.e. what is the *most probable neighborhood*, depends of the problem. In the example case the terms M=0,1,2,3 should be included because they cover 99% of all the mutants that will occur during the search.

Implementation of the Gene Bank

As the Gene Bank holds numerous chromosomes and each chromosome produced is first sought from the Gene Bank the accessing should be fast and efficient. This excludes a linear list with one entry, which would be obviously too slow to handle. On the other hand a unique address for each possible chromosome instance would be impossible to implement due to the combinatorial explosion and would also be unnecessary as recalling the most probable neighborhood concept.

Figure 3 illustrates the adopted structure, which is based on *hashing* technique well known in the computer science, e.g. [REF89], [Sed90]. The idea is to map the data vector to an address space covered by the *hash table*. The mapping is incomplete in a sense that data vectors will not get unique addresses but instead an index (a key) to a table that references to a list where the vector is to be included. In the Gene Bank context each item in the list holds space for both the chromosome and fitness value. The list is organized in circular form, each item contains also references to its predecessor and successor enabling access to the previous and next items.

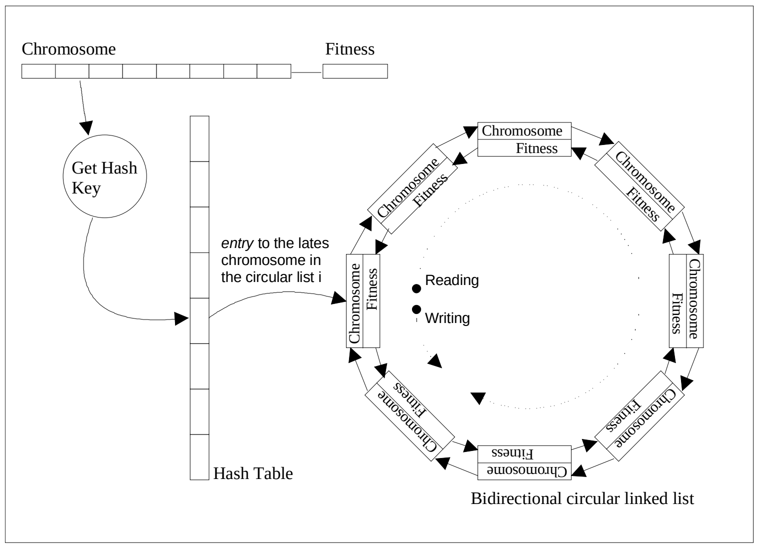


Figure 3. Principle of Gene Bank implementation

EvaluateChromosomes
begin
 	
repeat
 	
Key = GetHashKey(Chromosome)
if not ReadGeneBank(Key, Chromosome)
begin
EvaluateFitness(Chromosome)
WriteGeneBank(Key, Chromsome)
end
 	
until AllChromsomesEvaluated
 	
end
	WriteGeneBank(Key, Chromosome)
	begin
	
	Entry = HashTable[Key]->Prev
	Entry->Chromosome = Chromosome
	Entry->Fitness = Fitness
	
	HashTable[Key] = Entry
	
	end
		ReadGeneBank(Key, Chromosome)
		begin
		
		Entry = HashTable[Key]
		Item = Entry
		
		while (Item->Next <> Entry) and
		(Chromsome<>Item->Chromsome)
		begin
		Item = Item->Next
		end
		
		if(Chromosome = Item->Chromsome)
		begin
		Fitness = Item->Fitness
		return TRUE
		end
		return FALSE
		
		end

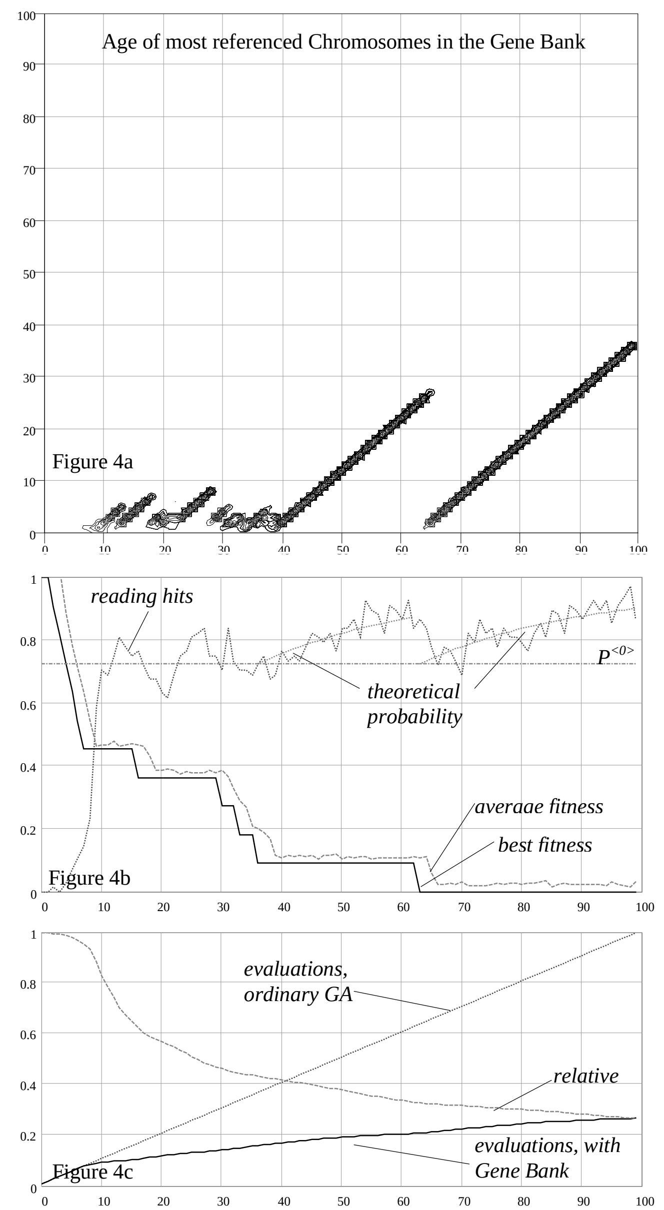
Table 2. Pseudo-Codes for the Gene Bank operations

The presented Gene Bank structure lends itself naturally to the epochal behavior of the genetic search. The space is allocated only for the most probable neighborhood around the current S_0. The older chromosomes belonging to other subbasins are displaced automatically by the properties of the circular lists. Table 2 gives pseudo-codes for the Gene Bank operations.

The proper operation of the hashing algorithm is important. If the hash function does not generate evenly distributed hash keys some of the lists will be overloaded while others remain empty. This yields to increased access times and even worse, to loss of the already evaluated chromosomes at present subbasin. A complete survey of hashing algorithms can be found from [Jen97].

Test problem and results

To proof the presented theory a simple string search problem where GA searches a 16-characters alphabet string was implemented. The fitness value is simply the number of wrong characters in the string, which is being minimized. Because there may be *exactly* the same fitness value with dissimilar chromosome instances the selection process and convergence is assisted with summing of negligible random perturbation to the fitness values. This assistance is actually a property of Gene Bank - normally a noisy fitness function impairs convergence [MG95]. The GA-parameters were: N_p = 80, G = 16, r = 26, p_m = 0.02, λ = 0.15 and the number of generations N_G = 100.



Figures 4a to 4c illustrate a typical search with the Gene Bank on the test problem. In 4b and 4c the y-axis values are normalized to range 0..1 while x-axis is the generation. The curve *reading hits* in 4b is the number of successful references to the Gene Bank, i.e. the reading attempt when the chromosome being evaluated can be found from the Gene Bank. The *reading hits* curve is the ratio of *successful readings* divided by *all readings*. The curve *theoretical probability* is calculated from formula (9). Two regions are fitted in: epochs between generations 36..63 and 64..99. The *theoretical probability* predicts considerably well the actual hits from the Gene Bank in these epochs. Between generations 10..35 the epochs are so short that regularity in the behavior is difficult to observe. However, after generation 10 the mutation order M=0 is dominant. This can be seen in *reading hits* which fluctuates around the $P^{<0>}$ = 0.724. Before generation 10 the population is so diverse that just a few chromosomes can be accessed from the Gene Bank.

Figure 4c illustrates the number of evaluations needed with ordinary GA compared to GA with Gene Bank. The number of evaluated chromosomes increases linearly in the ordinary GA. The evaluations in the GA with Gene Bank follow the ordinary curve for the first 10 generations as population was diverse during the very first generations. After that the margin begins to increase and in the end of the search the Gene Bank version has performed only approximately 30% of the fitness evaluations needed in the ordinary GA.

Figure 4a illustrates the age of the most frequently referenced chromosomes in the Gene Bank. It indicates clearly how new chromosomes appear to the Gene Bank when a new subbasin is reached. Those chromosomes are referenced until a new epoch begins and the chromosomes in that subbasin conquer the Gene Bank.

The number of generations (N_G =100) in the example was empirically selected so that the search most often converges to the final solution. We saw that Gene Bank is ineffective for the very first generations of the search. On the other hand, after that the *reading hits* stays most of the time above the line $P^{<0>}$. Those areas below and above $P^{<0>}$ approximately compensate each other. A rule of a thumb may be stated: if the number of generations is such that GA usually converges to the solution of the problem then the number of evaluations required is roughly

$$\tilde{N}_E \approx N_p \cdot N_G \cdot (1\text{-}P^{<0>}) \qquad (10).$$

In fact (10) gives an estimate of the number of the *unique* chromosomes arisen during the entire genetic search. Formula (10) can be used also to estimate the effectiveness of the Gene Bank to the optimization problem at hand when the fitness function complexity is known.

The operation of hash table and the importance of the well behaving, evenly distributing hash function is seen in figure 5.

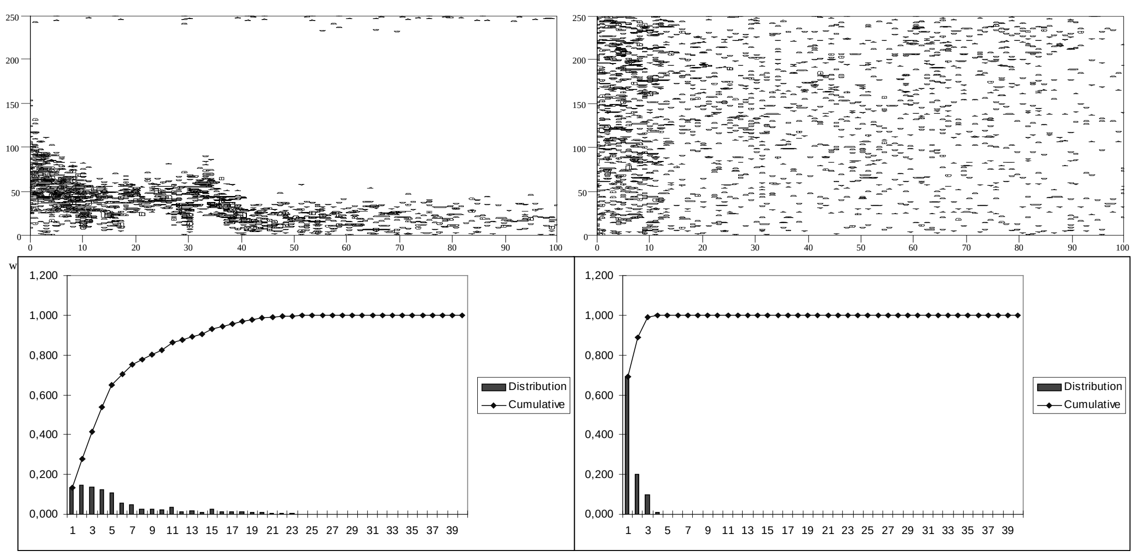


Figure 5. Two hashing functions

The hash function in the left is not able to map the chromosomes evenly to the hash table during the search. This concentrates the chromosomes into a few keys and the occupied lists contain lots of data with drawbacks mentioned above. The hash function on the right has even distribution. That yields to less occupied lists, on the average over 99% of the accessed chromosomes can be found from the 3 first list items.

Discussion

The concept of Gene Bank was presented with a simple string search problem. Originally the Gene Bank was developed for a landing call allocation problem faced when a group of elevators is controlled to work together [TY99]. In that NP-hard real time control application the problem is to find optimal routes for the elevators so that waiting times of passengers are minimized. The fitness function is a computationally complex simulation model of the elevator group. The response time, which has to be apparently below 0.5 seconds, is almost linearly proportional to the number of fitness evaluations. In this case the Gene Bank is able to reduce the response time on the average with 65% being a key technique in the implementation of the genetic call allocation algorithm.

A general way to estimate the number of evaluations during genetic search is to multiply population size by the number of generations [Gol99]. However, according to results presented in this paper, a majority of candidate solutions created during the course of genetic search are not unique. Therefore we suggest using formula (10) instead to estimate the number of unique chromosomes arising during the search. The ordinary GA can not distinguish between unique and already arisen chromosomes but GA enhanced with the Gene Bank evaluates only the necessary solution alternatives. Depending of the problem the reduction in the number of evaluations and search time may be substantial.

Quite often the methods presented in GA-literature to improve genetic search assume certain properties of the application or then application specific properties are incorporated into the GA itself or the method may posses a set of parameters that need to be optimized for current problem at hand. However, apart from that, the Gene Bank technique is a systematic, integral part of GA and entirely invisible to the application. The only parameter that needs to be taken care is the Gene Bank's storage capacity, which can be easily defined as presented earlier. Furthermore, with the memory capacity of present computers the allocated memory may be selected in a safe way – just let it be large enough.

Another aspect is that Gene Bank makes GA more robust in a sense of the number of generations - in the end of the search most of the chromosomes are accessed quickly from the Gene Bank. So the number of generations may be selected more freely without linearly increasing the search time in the computationally demanding problems.

Conclusions

The concept of Gene Bank was introduced. Behavior of population and Gene Bank was considered both theoretically and experimentally during an epoch the search remains in a subbasin. The results indicate that the adopted approach to model the behavior of population and the Gene Bank is appropriate.

Gene Bank is a systematic and entirely application independent method to improve the performance of a binary coded Genetic Algorithm in computationally demanding applications. Depending of the fitness function complexity the improvement in search time may be substantial. A class of 65% improvement in the response time has been obtained in a real world real time control problem where landing calls given by passengers in a building are allocated to elevators in optimal way.

References

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

[Gol99]
Goldberg, David E. (1999). Using Time Efficiently: Genetic-Evolutionary Algorithms and the Continuation Problem. IlliGAL Report No. 99002, Department of General Engineering, University of Illinois at Urbana-Champaign.
Available at http://www-illigal.ge.uiuc.edu/orderform.html

[Jen97]
Jenkins, Bob (1997). Hash Functions. Dr. Dobb's Journal, September 1997, pp. 107-109.

[MG95]
Miller B.L. and Goldberg D.E.
Genetic algorithms, selection schemes and the varying effects of noise. IlliGAL Report No.95009, November 1995. Department of General Engineering, University of Illinois at Urbana-Champaign.

[REF89]
Reilly, D. Edwin and Federighi, D. Francis
Pascal Algorithms, a Pascal-Based introduction to computer science
Houghton Mifflin Company, Boston, 1989.

[Sed90]
Sedgewick, Robert
Algorithms in C
Addison-Wesley Publishing Company, 1990.

[TY99]
Tyni, Tapio and Ylinen, Jari
Procedure for allocating landing calls in an elevator group.
Australian Patent No. AU698715, dated 18 February 1999.

[vNC98]
van Nimwegen, Erik, and Crutchfield, James P. (1998). Optimizing Epochal Evolutionary Search: Population-Size Independent Theory. Santa Fe Institute Working Paper 98-06-046. To appear in Computational Mechanics and Engineering of the Journal Computer Methods in Applied Mechanics and Engineering, D. Goldberg, editor.
Available at http://www.santafe.edu/sfi/publications/working-papers.html

[Whi89]
Whitley, Darrel (1989). The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In Schaffer, J.D. ed., Proceeding of the Third International Conference on Genetic Algorithms, pp. 166-121. Morgan Kaufmann Publishers, Inc., San Meteo, CA.