A PROBABILISTIC SEARCH ALGORITHM FOR FINDING OPTIMALLY DIRECTED SOLUTIONS

B. Raphael and I.F.C. Smith
Institute of Structural Engineering and Mechanics (ISS-IMAC)
EPFL-Federal Institute of Technology
CH-1015 Lausanne
Switzerland
Email : Benny.Raphael@epfl.ch

ABSTRACT: A new global search technique called "Probabilistic Global Search-Lausanne" (PGSL), is presented in this paper. The technique is based on selective sampling of the search space according to a probability distribution function. The performance of the technique is compared with genetic algorithms using non-linear benchmark problems involving a large number of variables. For most of these problems, PGSL performs better than GAs. PGSL shows promise for improving the performance of advanced software for the construction industry.

KEYWORDS: Global search, genetic algorithms, optimisation

1. INTRODUCTION

Evolutionary search techniques such as Genetic Algorithms (GA) have recently gained considerable attention. They have been used for solving a wide range of problems including function optimisation and learning. They are applied mostly to exponentially complex problems in which it is impossible to obtain exact solutions within reasonable amount of time.

In this paper, a new global search technique, called "Probabilistic Global Search-Lausanne" (PGSL), is presented. The technique is based on selective sampling of the search space according to a probability distribution function that is varied dynamically during the search process. Probabilities are increased in regions where good solutions are found and hence these regions are searched with greater intensity. Similar to many other methods derivatives are not used and therefore, problems due to numerical instability are avoided and the chance of getting trapped in local optima is reduced. Results of benchmark tests indicate that this technique performs better than genetic algorithms on a wide range of problems. Finally, several potential applications for the AEC industry are discussed.

2. SEARCH TECHNIQUES

There are several classes of engineering problems that require searching in large solution spaces. Typically, search methods involve generating solutions and testing them for constraint satisfaction and other optimality criteria. Since the search space is huge in most engineering situations, it is impossible to perform an exhaustive search. Probabilistic approaches have been used where a near optimal solution is sufficient. In this section, some of these techniques and their applications are discussed.

2.1 Random start local search

Local search techniques involve iteratively improving upon a solution point by searching in its neighbourhood for better solutions. If better solutions are not found, the process terminates; the current point is taken as a locally optimal solution. Since local search performs poorly when there are multiple local optima, a modification of this technique has been suggested in which local search is repeated several times using randomly selected starting points. This process is computationally expensive; after each iteration, search starts from a point very far away from the optimum and no information obtained from previous iterations is reused.

Random Bit Climbing (RBC) [Davis 1991] is a form of local search in which neighbouring points are randomly evaluated and the first move producing an improvement is accepted for the next stage. This process continues till the point cannot be improved any further.

2.2 Heuristic search

This is another search technique in which problem dependent heuristics are used to guide the search path. A well-known example of this is the Lin-Kernighan heuristic for solving the travelling salesman problem [Lin 1973]. It is found that this heuristic reduces the computation time by more than ten-fold compared to other generic search methods such as simulated annealing [Martin 1995].

2.3 Simulated annealing

Simulated annealing [Kirkpatrick et.al. 1983] is a generic method based on Markov Chains. It improves upon local search algorithms by occasionally allowing movements to worse solution points and is thus capable of jumping out of local optima. The method draws analogy from the annealing of metals where the temperature controls the arrangement of atoms in their lowest energy configuration during the crystallisation process. In simulated annealing, moves are accepted or rejected with a certain probability depending on a function of the temperature such that at higher temperatures there is greater probability of accepting inferior moves. Temperature is gradually brought down so that the solution converges. One of the drawbacks of simulated annealing is that it works with point solutions rather than a class of "optimal" solutions. Nevertheless, point solutions might be useful in many cases.

2.4 Genetic algorithms

Genetic algorithms draw analogy from the evolution of species in biology. Species evolve by means of genetic operators such as crossover and mutation and they survive through the mechanism of the survival of the fittest. In genetic algorithms, this process is simulated by encoding potential solutions (individuals) using a chromosome-like data structure. Unlike local search and simulated annealing, genetic algorithms work with a population of potential solutions. New individuals (children) are created in the population through reproduction using crossover and mutation operators. These operators ensure that children inherit qualities of parents and they are passed on from one generation to the other. Only a certain number of good quality individuals survive each generation and this ensures that the quality of the population improves with each generation.

Over the years, several improvements have been suggested to the original algorithm introduced by John Holland [Holland 1975] and the result is a family of algorithms based on evolutionary principles. Several variations of GA are discussed in the following subsections.

2.4.1 The simple genetic algorithm

The original algorithm introduced by John Holland is sometimes known as the simple genetic algorithm (SGA). Its steps are given below:

1. Generate initial population
2. Evaluate the population using a fitness function
3. Create an intermediate population (mating pool) containing individuals that take part in reproduction to generate the next population
4. Apply reproduction operators crossover and mutation to create the next population
5. Repeat steps 2-4 until the termination condition is met

The selection of individuals to be placed in the mating pool is performed according to a probability function that depends on the fitness. According to this scheme, individuals having higher fitness might be replicated multiple times to be placed in the mating pool. Crossover is applied to randomly selected pairs of strings representing individuals according to a probability p_c. After crossover, a mutation operator is applied with a low probability p_m. The newly created individuals replace parents.

2.4.2 Steady state genetic algorithm

This is a variation of the simple genetic algorithm. The main differences are the following:

- There is no intermediate population. Two parents are selected for reproduction and the offspring is immediately placed into the population
- Offspring do not replace parents, but the least fit member of the population
- There is no fitness proportionate reproduction

For several classes of problems the steady state genetic algorithm (Syswerda 1991) performs much better than the simple genetic algorithm.

2.4.3 CHC

The CHC algorithm developed by Larry Eshelman [Eshelman 1991] is another variation of the genetic algorithm. CHC stands for Cross generational elitist selection, heterogeneous recombination (by incest prevention) and Cataclysmic mutation. Important features of the algorithm are given below:

- After recombination, N best individuals are selected from the parent and offspring to create the next generation. Duplicates are removed from the population.
- Individuals are randomly selected for reproduction, however, certain restrictions are imposed on which strings are allowed to mate. Strings within a certain hamming distance are not allowed to mate.
- A form of uniform crossover called HUX is used in which exactly half of the differing bits are swapped.
- When population converges and starts producing more or less same strings, cataclysmic mutation is activated. All strings except the best are heavily mutated.

Recent evaluations indicate that CHC is generally more efficient than SGA and steady state genetic algorithm.

3. PROBABILISTIC GLOBAL SEARCH LAUSANNE

The Probabilistic Global Search Lausanne (PGSL) algorithm was developed based on the observation that optimally directed solutions can be obtained efficiently through carefully sampling the search space without using special operators. The basic assumption is that better points are more likely to be found in the neighbourhood of good points. Hence, search is intensified in regions containing good solutions.

The search space is sampled by means of a probability distribution function (PDF) defined over the entire search space. Each axis is divided into a fixed number of intervals and a uniform probability distribution is assumed in the beginning. As search progresses, intervals and probabilities are dynamically updated so that points are generated with higher probability in regions containing good solutions. The search space is gradually narrowed down so that convergence is achieved.

The algorithm includes four nested cycles:

- Sampling
- Probability updating
- Focusing
- Subdomain

In the sampling cycle (innermost cycle) a certain number of points, n_s, are generated randomly according to the current PDF. Each point is evaluated by the user defined objective function and the best point is selected. In the next cycle, probabilities of regions containing good solutions are increased and probabilities decreased in regions containing less attractive solutions. In the third cycle, search is focused on the interval containing the best solution after a number of probability updating cycles, by further subdivision of the interval. In the subdomain cycle, the search space is progressively narrowed by selecting a subdomain of smaller size centred on the best point after each focusing cycle.

Each cycle serves a different purpose in the search for a global optimum. The sampling cycle permits a more uniform and exhaustive search over the entire search space than other cycles. Probability updating and focusing cycles refine search in the neighbourhood of good solutions. Convergence is achieved by means of the subdomain cycle.

3.1 Terminology

The following definitions are used in the explanation of the algorithm:

Search space: The set of all potential solutions. It is an n-dimensional space with an axis corresponding to each variable. The user defines the minimum and maximum values along each axis. A subset of the search space is called a subdomain.

Solution point: A point in the search space consisting of a set of values for each variable.

Axis width: The difference between the minimum and the maximum along an axis of the search space or a subdomain.

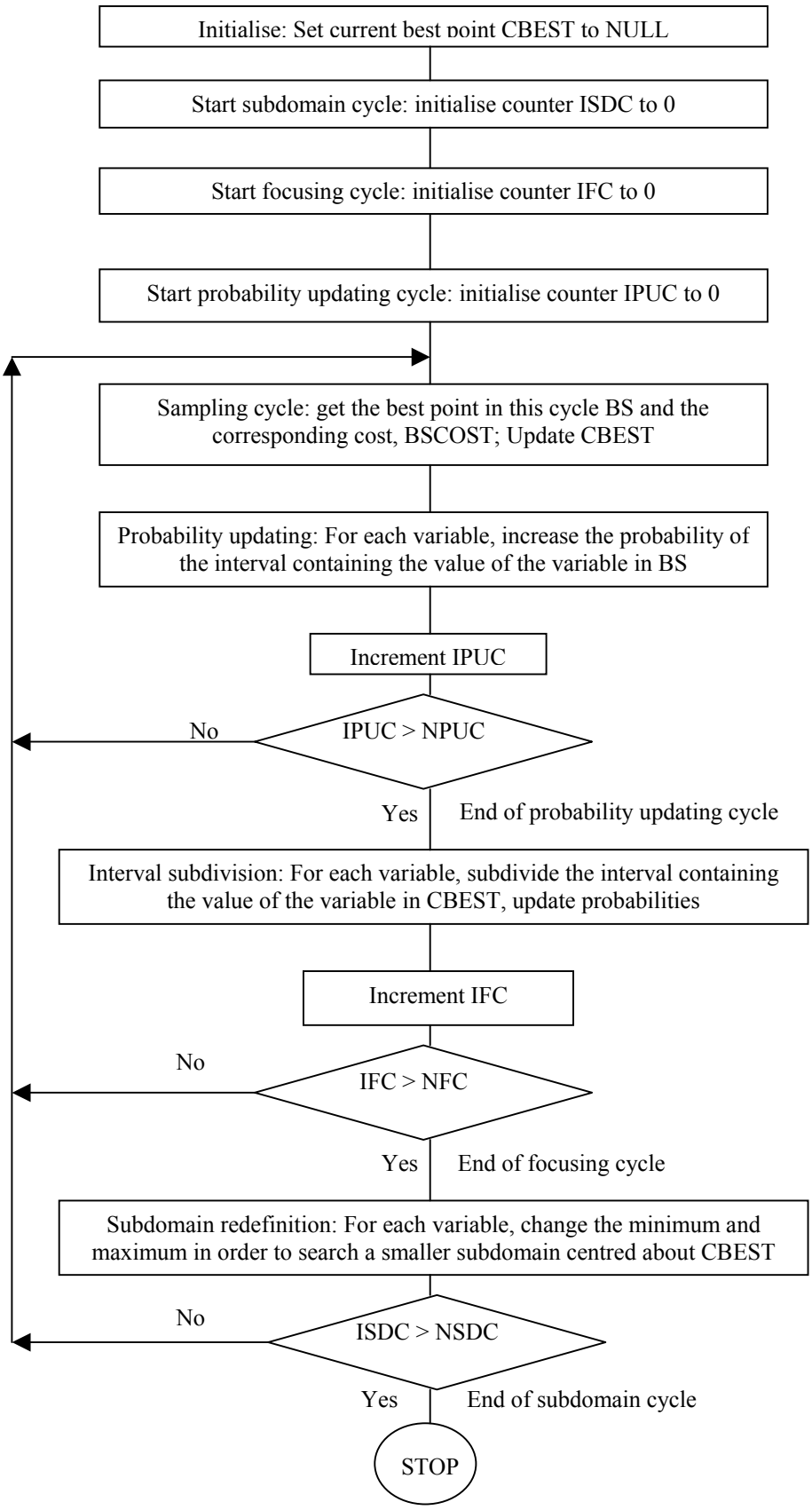

Figure 1. Flow chart for the PGSL algorithm

Cost function: A user-supplied function to evaluate a solution point. The value of the cost function for a given point is called the *cost* of the solution point.

Probability density function, PDF: The PDF of a variable is defined in the form of a histogram. The axis represented by the variable is discretised into a fixed number of intervals, *NINTERVALS*. Uniform probability distribution is assumed within each interval. The cumulative distribution function (CDF) is obtained by integrating the PDF.

Important parameters involved in the algorithm are listed below:

Number of samples, NS: The number of samples evaluated in the sampling cycle.

Iterations in the probability updating cycle, NPUC: The number of times the sampling cycle is repeated in a probability updating cycle.

Iterations in the focusing cycle, NFC: The number of times the probability updating cycle is repeated in a focusing cycle.

Iterations in the subdomain cycle, NSDC: The number of times the focusing cycle is repeated in a subdomain cycle.

Subdomain scale factors, SDSF1, SDSF2 : The default factors for scaling down the axis width in the subdomain cycle. SDF1 is used when there is an improvement and SDF2 if there is no improvement.

3.2 Algorithm details

The algorithm is illustrated in the form of a flowchart in Figure 1 and is explained in more detail below:

3.2.1 Initialisation

The search space is defined by reading the minimum and maximum values for each variable given by the user. The PDF of each variable is created by assuming a uniform distribution over the entire domain. All PDFs have intervals of constant width in the beginning.

3.2.2 Sampling cycle

NS points are generated randomly by generating a value for each variable according to its PDF. This is similar to sampling in the Montecarlo technique. Each point is evaluated and the point having the minimum cost, BS (Best Sample) is selected.

3.2.2 Probability updating cycle

The sampling cycle is invoked NPUC times. After each iteration, the PDF of each variable is modified using the probability-updating algorithm. This ensures that the sampling frequencies in regions containing good points are increased. The evolution of the PDF for a variable after several sampling cycles is illustrated in Figure 2.

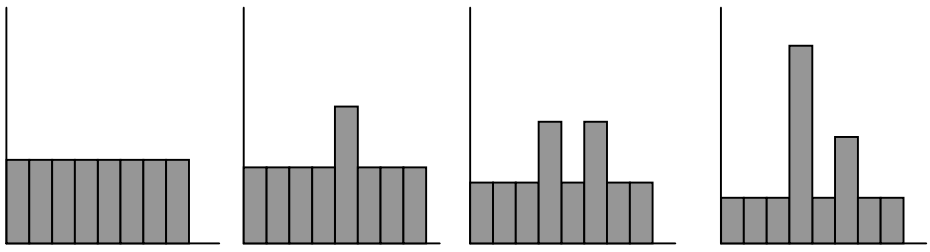

Figure 2: Evolution of the PDF of a variable after several probability updating cycles

3.2.3 Probability-updating algorithm

The PDF of a variable is updated through these steps:

- Locate the interval containing the value of the variable in BS.
- Multiply the probability of the interval by a factor (greater than 1), PUF.
- Normalise the PDF

3.2.3 Focusing cycle

The probability updating cycle is repeated NFC times. After each iteration, the search is increasingly focused on the interval containing the current best point, CBEST. This is done by subdividing the interval containing the value of each variable in CBEST. The evolution of the PDF after several probability-updating cycles is illustrated in Figure 3.

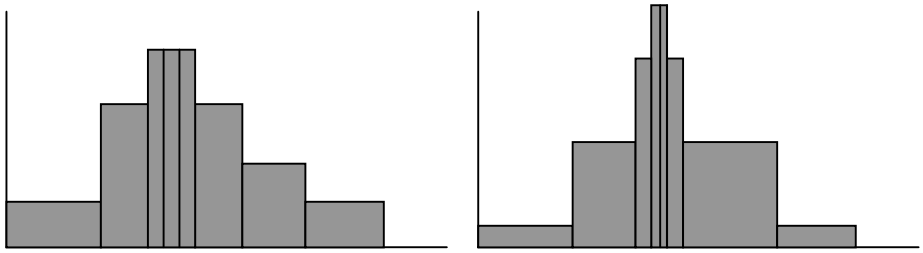

Figure 3: Evolution of the PDF of a variable after several focusing cycles

3.2.4 Interval subdivision

The following steps are used for subdividing intervals in the focusing cycle:

- Locate the interval (called BESTINTERVAL) containing the value of the variable in CBEST.
- Divide the interval into NDIV uniform subintervals.
- Assign 50% probability to BESTINTERVAL, (so that half of the points generated will be in this interval). Divide this probability uniformly to its subintervals.
- Calculate the number of intervals into which the remainder of the domain should be divided so that the total number of intervals remain constant.
- Distribute the remaining probability to the region outside the BESTINTERVAL so that the PDF decays exponentially away from the BESTREGION.

After subdivision, intervals no longer have the same width and probabilities are heavily concentrated near the current best.

3.2.5 Subdomain cycle

In the subdomain cycle, the focusing cycle is repeated NSDC times and at the end of each iteration, the current search space is modified. In the beginning, the entire space (the global search space) is searched, but in subsequent iterations a subdomain is selected for search. The size of the subdomain decreases gradually and the solution converges to a point. A subdomain is selected by changing the minimum and maximum of each variable.

While choosing the next subdomain, certain precautionary measures are taken to avoid premature convergence. Firstly, a higher scale factor is used after an iteration

that does not produce a better cost. This avoids rapid reduction of the axis width after several unsuccessful iterations. Secondly, the statistical variations of the values of the variable in previous iterations are considered in determining the new minimum and maximum. If the value of the variable fluctuates by a large amount the convergence is slowed down.

The method to compute the new values of minimum and maximum for each variable is explained in pseudo-code below:

- Let XP = the value of the variable in CBEST
- Let DX = (Current Axis Width)/2
- Let GX1 = Minimum of the axis in the global search space
- Let GX2 = Maximum of the axis in the global search space
- Let STDEV be the standard deviation of the value of the variable in previous 5 iterations
- If there has been an improvement in cost in the current iteration, scale factor, SCF = SDF1, else SCF = SDF2.
- The new half width, NDX = DX * SCF.
- If NDX < STDEV NDX = STDEV
- The new minimum of the axis, X1 = XP-NDX.
- The new maximum of the axis X2 = XP+NDX.
- If X1 < GX1 then X1 = GX1
- If X2 > GX2 then X2 = GX2

3.3 Choosing values for parameters

Values of parameters that have been empirically found to be insensitive to the problem-type are given below:

- Number of intervals in the PDF, *NINTERVALS* = 20
- The number of subintervals, NDIV = 6
- Subdomain scale factor SDSF2 = 0.96

Problem dependent parameters include:

- Number of samples, NS
- Iterations in the probability updating cycle, NPUC.
- Iterations in the focusing cycle, NFC
- Iterations in the subdomain cycle, NSDC
- Subdomain scale factor, SDSF1

It is found that for reasonably smooth problems, the values of NS and NPUC can be taken as 2 and 1 respectively. Increasing these values appear to produce no better results. However, for highly irregular domains higher values should be used. It can be shown that best results are obtained when these values are proportional to the number of reasonably regular sub-regions within the space. However, even for highly non-linear problems, the default values of 2 and 1 seem to work quite well and they were used in all the benchmark problems listed in the next section.

The value of NFC could be between 10N and 20N, where, N is the number of variables in the problem. A higher value results in more intensive search in the neighbourhood of the current best point.

The value of SDSF1 should be between 0.5 and 0.99. A lower value results in rapid reduction in the sizes of subdomains and may cause premature convergence. A higher value slows down convergence and it may take much longer to find the optimum,

however, much better quality solutions are obtained. The following empirical formula is found to produce good results:

$SDSF1 = N^{(-1/N)}$

The value of NSDC controls the precision of results and is dependent on the scale factors. A lower value results in the axis width of the subdomain very large after all iterations. The length of search (the number of evaluations) can be modified by adjusting the values of SDSF1 and NSDC.

4. BENCHMARK TESTS

The performance of PGSL is evaluated by testing it on several benchmark problems. Recent publications [Borkowski 1999, Topping 1999] indicate that genetic algorithms are being more widely used for solving general optimisation problems compared to other generic methods. Hence, the performance of PGSL is compared with three versions of GAs.

De Jong [De Jong 1975] first proposed common test functions (F1-F5) with multiple optima to be used for evaluating genetic algorithms. However, it has been shown that some of them can be treated satisfactorily by local search [Davis 1991]. More difficult test functions have been proposed recently [Whitley 1995], which are highly nonlinear and which can be scaled to a large number of variables. Some of these functions are used for testing the performance of the PGSL algorithm. A short description of the test functions are given below:

F8 (Griewank's function):
It is a scalable, nonlinear, and non-separable function given by

$$f(x_{i|i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (\cos(x_i / \sqrt{i}))$$

Expanded functions:
Expanded functions [Whitley 1995] are constructed by starting with a primitive nonlinear function in two variables, F(x,y), and scaling to multiple variables using the formula,

$$EF(x_{i|i=1,N}) = \sum_{j=1}^{N} \sum_{i=1}^{N} F(x_i, x_j)$$

The expanded functions are no longer separable and introduce nonlinear interactions across multiple variables. An example is the function EF10 created using the primitive function F10 given by the following equation:

$$F10(x, y) = (x^2 + y^2)^{0.25} \left[\sin^2(50 (x^2 + y^2)^{0.1}) + 1\right]$$

Composite functions:
A composite function can be constructed from a primitive function $F(x_1, x_2)$ and a transformation function T(x,y) using the formula

$$EF\ (x_{i|i=1,N}) = F(T(x_n, x_1)) + \sum_{i=1}^{N-1} F(T(x_i, x_{i+1}))$$

The composite function $EF8_{avg}$ is created from the *Griewank's function, F8* using the transformation function T(x,y) = (x+y)/2

The composite test function $EF8_{F2}$ is created from the *Griewank's function, F8* using the De Jong function F2 as the transformation function. F2 is defined as

$$F2(x, y) = 100\ (x^2 - y^2) + (1 - y^2)$$

The composite functions are known to be much harder than the primitive functions and are resistant to hill climbing.

4.1 Results of benchmark tests

The performance of PGSL is compared with results reported for three programs based on genetic algorithms, namely, ESGAT, CHC and Genitor [Whitley 1995]. ESGAT is an implementation of simple genetic algorithm and Genitor an implementation of steady state genetic algorithm. Four test functions are used for comparison, F8, EF10, $EF8_{AVG}$ and $EF8_{F2.}$ All these test functions have a known optimum (minimum) of zero. It is known that [Whitley 1995] local search techniques perform poorly in solving these problems.

Results are summarised in tables 1-4. Thirty trial runs were performed for each problem using different seed values for random numbers. In each trial, a maximum of 500,000 evaluations of the objective function is allowed. Performance is compared using three criteria.

1. The success rate (the number of trials in which the global optimum was found)
2. The mean solution obtained in all the trials. The closer the mean solution is to zero (the global optimum) the better is the performance of the algorithm.
3. The mean number of evaluations of the objective function required to obtain the global optimum (only for trials in which the optimum was found)

4.1.1 Simple F8 test function

Results for simple F8 test function are given in Table 1. Thirty trial runs were performed on problems with 10, 20, 50 and 100 variables. PGSL has a success rate of 100% for 50 and 100 variables, no version of GA is able to match this. (Surprisingly, the success rate is slightly lower for fewer variables). However, the mean number of evaluations to obtain the optimum is higher than CHC and Genitor for this problem.

Num. Variables		**10**	**20**	**50**	**100**
Succ	ESGAT	6	5	0	0
	CHC	30	30	29	20
	Genitor	25	17	21	21
	PGSL	28	29	30	30
Mean solution	ESGAT	0.0515	0.0622	0.0990	0.262
	CHC	0.0	0.0	0.00104	0.0145
	Genitor	0.00496	0.0240	0.0170	0.0195
	PGSL	0.0007	0.0002	0.0	0.0

Mean Num. Evals.	ESGAT	354422	405068		
	CHC	51015	50509	182943	242633
	Genitor	92239	104975	219919	428321
	PGSL	283532	123641	243610	455961

Table 1: Results for Simple F8 test function

4.1.2 EF10 test function

Results for the extended function EF10 are summarised in Table 2. It can be seen that PGSL has a very high success rate (27 out of 30) even for 50 variables. For all criteria, PGSL performs better than all versions of GAs.

Nb Var		**10**	**20**	**50**
Succ	ESGAT	25	2	0
	CHC	30	30	3
	Genitor	30	4	0
	PGSL	30	30	27
Mean solution	ESGAT	0.572	1.617	770.576
	CHC	0.0	0.0	7.463
	Genitor	0.0	3.349	294.519
	PGSL	0.0	0.0	0.509639
Mean Num Evals.	ESGAT	282299	465875	
	CHC	51946	139242	488966
	Genitor	136950	339727	
	PGSL	61970	119058	348095

Table 2: Results for the extended function EF10

4.1.3 $EF8_{AVG}$ test function

Results for the composite function $EF8_{AVG}$ are summarised in Table 3. For 20 and 50 variables, none of the algorithms is able to find the exact global optimum. For 10 variables the performance of CHC is comparable with that of CHC. In terms of the mean value of the optimum, PGSL outperforms all other algorithms.

Nb Var		**10**	**20**	**50**
Succ	ESGAT	0		
	CHC	10		
	Genitor	5		
	PGSL	9		
Mean solution	ESGAT	3.131	8.880	212.737
	CHC	1.283	8.157	83.737
	Genitor	1.292	12.161	145.362
	PGSL	0.0151	0.1400	1.4438
Mean Num Evals.	ESGAT			
	CHC	222933		
	Genitor	151369		
	PGSL	212311		

Table 3: Results for $EF8_{AVG}$ test function

4.1.4 $EF8_{F2}$ test function

Results for the composite function $EF8_{F2}$ are given in Table 4. None of the algorithms is able to find the global optimum for this problem. However, in terms of the quality of the mean solution, PGSL fares better than the rest.

Nb Var		**10**	**20**	**50**
Mean solution	ESGAT	4.077	47.998	527.1
	CHC	1.344	5.63	75.0995
	Genitor	4.365	21.452	398.12
	PGSL	0.123441	0.4139	1.6836

Table 4: Results for $EF8_{F2}$ test function

4.2 Discussion

Among the three implementations of GAs considered in this section, CHC performs better than the rest. In most cases, the quality of results produced by PGSL is better than CHC. However, PGSL requires a greater number of evaluations than CHC, especially for small problems. The overall performance of PGSL is comparable to, if not better than CHC.

5. POTENTIAL FOR APPLICATIONS IN CONSTRUCTION

There is potential for the application of the technique in construction. Several applications of global search techniques can be found in recent publications. For example, simulated annealing has been used for the control of tensegrity structures [Smith and Shea 1999]. GAs have been widely used in design and optimization [Sisk et.al.1999, Grierson and Khajehpour 1999]. An application of PGSL for bridge diagnosis is presented in these proceedings [Robert-Nicoud et.al. 2000].

6. CONCLUDING REMARKS

Results reported in this paper indicate that PGSL is a robust search technique that performs well in spaces with multiple local optima. For test problems of continuous non-linear functions with multiple local minima, its performance is better than GAs in terms of i) the quality of solutions, ii) the success rate as well as iii) the number of evaluations required to find the optimum. PGSL is based on the idea of selective sampling and evaluation of the search space and therefore, it has less computational overhead than genetic algorithms.

PGSL resembles grid search techniques in which the space is partitioned and refined over time. However, grid search techniques suffer from exponential growth of partition elements resulting in exponential growth of samples to cover all partitions effectively and consequently, large tables to maintain them. PGSL avoids this problem by creating a single PDF for each variable. Interesting partitions are encoded implicitly in this PDF and hence the complexity of the algorithm is linear with respect to the number of variables that define the search space.

Comparison of the method with other search techniques and testing on large scale engineering examples are currently in progress.

ACKNOWLEDGEMENTS

This research is funded by the Alliance for Global Sustainability (AGS), Swiss National Science Foundation (NSF) and the commission for technology and innovation (CTI). We would like to thank Dr. K. De Jong and K. Shea for valuable comments and suggestions. We would also like to thank Logitech SA and Silicon Graphics Incorporated for supporting this research.

REFERENCES

Borkowski A. (1999). Artificial Intelligence in Engineering, Information Technology for design, manufacturing, maintenance and monitoring, Politechnika Warswawska.

Davis L. (1991). Bit-climbing, representational bias and test suite design, In L.Booker and R.Belew, ed., Proceedings of the 4th international conference on GAs, Morgan Kauffman.

De Jong, K (1975). Analyis of the behaviour of a class of genetic adaptive szstems. Ph.D. thesis, Univerisity of Michigan, Ann Arbor.

Eshelman L. (1991). The CHC adaptive search algorithm. Foundations of genetic algorithms, G.Rawlins (editor), Morgan-Kaufmann. Pp. 256-283.

Grierson D.E., S.Khajehpour (1999), Multi-criteria conceptual design of office buildings using adaptive search, In Artificial intelligence in engineering, Information technology for design, manufacturing, maintenance and monitoring, Adam Borkowski (editor), Politechnika Warswawska.

Holland J. (1975). Adaptation in natural artificial systems, University of Michigan Press.

Kirkpatrick, S.,C.Gelatt and Vecchi M. (1983). Optimisation by simulated annealing, Science. pp. 220:673.

Kin S., Kernighan B. (1971). An effective heuristic for the travelling salesman problem, Operations research, 21:498.

Martin O. (1995). Combining simulated annealing with local search heuristics, Metaheuristics in combinatoric optimization, G.Laporte and I.Osman (editors).

Y.Robert-Nicoud, B.Raphael, Ian Smith (2000). Decision support through multiple models and probabilistic search, In proceedings of Construction Information Technology 2000, Iceland building research institute.

Sisk G.M., Moore C., Miles J. (1999). A decision support system for the conceptual design of building structures using a genetic algorithm, In Artificial intelligence in engineering, Information technology for design, manufacturing, maintenance and monitoring, Adam Borkowski (editor), Politechnika Warswawska.

Smith, I. and Shea, K. (1999). Extending active control to build intelligent structures, Structures for the Future - The Search for Quality, IABSE Reports, Vol. 83, International Association for Bridge and Structural Engineering, Zurich, pp 1057-1064.

Syswerda G. (1991). A study of reproduction in generational and steady-state genetic algorithms, Foundations of Genetic algorithms, G.Rawlins, editor, Morgan-Kaufmann. pp.94-101.

Topping B. and Kumar B. (1999). Optimization and control in civil and structural engineering, CIVIL-COMP Press.

Whiltley D. (1995). Building better test functions, In L. Eshelman, editor, Proceedings of the 6th international conference on GAs, Morgan Kauffman.