

Using PVM for Hunting Snake-In-The-Box Codes

M. Juric

Department of Computer Science and Information Systems
DePaul University
Chicago, Illinois 60604-2302

W.D. Potter, M. Plaksin
Artificial Intelligence Center
University of Georgia
Athens, Georgia 30602-7415

E-mail: `cphdmlj@hawk.depaul.edu`

June 25, 1996

Abstract

An In-coil in Q_n , the n -dimensional unit cube, is a simple cycle C in Q_n such that C has no chords in Q_n . An In-snake is a simple (open) path S in Q_n which has no chords in Q_n . The problem of finding a snake of maximum length suffers from severe combinatorial explosion. This paper describes the use of a Genetic Algorithm for finding snakes, and how this code was interfaced with the software package PVM (Parallel Virtual Machine), allowing us to adapt GA code written for single processor machines for use on a cluster of single processor machines acting in parallel.

1 Introduction

A snake is an induced subgraph of an n -dimensional hypercube which is a path. The problem of finding a snake of maximum length suffers from severe combinatorial explosion, and often taxes a single-processor machine to its limits. This paper describes the use of a Genetic Algorithm for finding snakes, and how this code was interfaced with the software package PVM (Parallel Virtual Machine), allowing us to adapt GA code written for single processor machines for use on a cluster of single processor machines acting in parallel.

Through the PVM package we are able to implement serial algorithms that run faster by distributing the computational task and resource burden among several machines. This package also enables us to implement parallel algorithms by coordinating processes that are operating on other machines, and allowing access to these processes as if they were executing on a single, parallel machine.

2 PVM

PVM is a software package that allows the programmer to transparently access heterogeneous computational platforms as a single concurrent machine [?]. The package consists of two parts: a daemon process that is user-configured to handle signalling tasks among designated machines, and a software library of routines to handle communication between machines and processes.

The simplest method for initiating a PVM sessions is to start the PVM daemon with a "hostfile" as an argument. This hostfile should contain information about the machines that PVM is to consider part of its virtual configuration: names or IP addresses, login names, passwords, etc. The PVM daemon then starts

on each of these machines and waits for messages. PVM can also be started in console mode. This allows the user to interactively start subtasks, collect statistics on running tasks, and perform other initiating and monitoring operations.

To use PVM, one must first spawn a PVM task. When a task is spawned, it is assigned a unique task identifier or tid which is used to communicate with that task. Communication takes the form of signalling or messaging. Signalling a task is accomplished through the use of standard POSIX-compliant system signals. Messaging uses the PVM daemon and library calls to pass user specified messages and signals of arbitrary length and meaning among tasks. Messages may be in the form of one or more basic C or Fortran data types. This means that complex data types such as structures must have their components packed and messaged individually.

In a most general PVM communication paradigm, a “parent” process spawns one or more “child” processes, communicating with them through messages or signals until the child process is either finished executing, or is terminated by the parent process.

3 GA

Genetic algorithms are heuristic search methods based on the notion of the survival of the fittest. They have been applied to a wide variety of problems, for example: Multiple Fault Diagnosis (MFD) [?, ?, ?, ?, ?, ?]; Set Covering (SC) and Traveling Salesman Problems (TSP) [?]; communication network configuration [?]; and control of natural gas pipelines and game playing [?]. Genetic algorithms are a type of stochastic search method, and are applied to NP-Hard problems in many areas ranging from scheduling optimization to designing optimal configurations and layouts. Developed by John Holland in 1975 [?], they are modeled from and mimic the theory of natural evolution.

The evaluation of a chromosome is done to test its “fitness” as a solution. Choosing and formulating an appropriate objective function is crucial to the efficient solution of any given genetic algorithm problem. In our case, searching for long paths, a fitness based on the length of the path found so far is a very good function.

4 Snakes

An In-coil in Q_n , the n-dimensional unit cube, is a simple cycle C in Q_n such that C has no chords in Q_n . That is, every edge of Q_n which joins two nodes of C is in fact an edge of C. The nodes of Q_n are the 2^n n-tuples of binary digits, and two nodes are joined by an edge of Q_n if they differ in exactly one coordinate. An In-snake is a simple (open) path S in Q_n which has no chords in Q_n . In other words, an n-snake is a path in Q_n having adjacencies only between nodes which are consecutive in the path.

N-dimensional cubes, or hypercubes have long been studied for their relevance to coding theory [?], and more recently due to the advent of parallel computing systems with hypercube communication topologies. Coils (or closed snakes) in hypercubes have received the most attention in the literature [?, ?]. Whether open or closed, snakes in Q_n have various applications, such as error-detection in analog-to-digital conversion [?]. In this case, the longer the snake, the more accurate the conversion will be. Additionally, snakes are related to algorithms used for disjunctive normal form simplification and for electronic combination locking schemes; again, the longer the snake, the more useful it is in the application [?]. Finally, the length of the longest snakes corresponds to the worst-case number of iterations in local-search algorithms [?]. Consequently, great interest has developed in determining the maximum length of open snakes, s_n , and coils (closed snakes), c_n , among all snakes in Q_n ; see for example [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?]. However, finding long snakes is such a difficult problem that the maximum lengths of snakes were previously known only for dimensions up to 6. Above dimension 6, only upper and lower bounds have been reported in the literature.

5 GA Experiment Setup

Our genetic algorithm (GA) experiments take two different tacks based on the representation of a snake. In one, we simply use the hypercube coordinates (node numbers in decimal that correspond to the binary encoding) as the representation of individuals in the population. For example, assume that the node sequence of the coil is: 1, 3, 7, 15, 14, 12, 8. We set the length of an individual to a constant (e.g., to 60 for Q_7) and then create the initial population at random. The first node is randomly chosen from among all nodes. Subsequent nodes in the individual are selected at random from the appropriate nodes in the adjacency table (in Q_7 for example, we have a table with 128 entries representing each node and the corresponding 7 nodes adjacent to them). This insures that individuals in the initial population are valid paths. The fitness of each individual is based on the cube of the length of the longest sub-snake in each.

Our other representation approach is based on the transition sequence [?, ?] of a path. The transition sequence is related to the single bit position that changes from one node to the next adjacent node. For example, the transition value from node 3 (0000011) to node 7 (0000111) is 3, and the transition value from node 6 (0000110) to node 14 (0001110) is 4. The transition sequence for the above coil is then: 1, 2, 3, 4, 1, 2, 3, 4. Again, each individual in the initial population is created at random and is insured to be a valid path. Insuring valid paths is much easier with this approach since the need for the adjacency table (or an adjacency calculation) is eliminated. Also, the fitness is the cube of the longest sub-snake in the sequence.

When using integer sequences that are order-based for individuals in the genetic algorithm, standard bit string genetic operators typically are inappropriate. Searching for snakes or traveling salesman routes demand that the operators that impact the order of the sequence maintain valid ordering. We use the enhanced edge recombination operator [?] with our node sequence representation because it focuses on maintaining node adjacency, an important feature for evolving long snakes. In addition, we use two variations of the evolutionary strategy. Namely, with the node sequence approach we use the standard mate selection and mating scheme where an entire next generation is created. We use the “one-at-a-time” replacement scheme used in *GENITOR* [?] for the transition sequence approach. In the one-at-a-time scheme, a new generation contains the same individuals as the previous generation except for the replacement of the worst (least fit) individual with the offspring of one mating pair of individuals selected in the usual fashion.

As for the specific GA parameter settings for the node sequence approach, we used a variety of population sizes ranging from 10,000 to 25,000, and a variety of crossover probabilities ranging from 0.6 to 0.95 with the enhanced edge recombination crossover scheme. We used a seeding scheme whereby we seeded the initial population of a trial with the best set of results from previous trials, although no fitness scaling was used. A trial ran for a specified number of generations and was then stopped. The limit here was in the neighborhood of 50,000 generations.

For the transition sequence approach, we used the same ranges for population size and crossover probabilities (i.e., 10,000 to 25,000, and 0.6 to 0.95). Mutation probabilities ranged from 0.01 to 0.04. We used a two-point crossover scheme with this approach. Again, seeding was used as with the node sequence approach. A trial was terminated whenever the population showed signs of convergence as indicated by no improvement in the best individual over several generations.

6 PVM Implementations

Our first implementation used the genetic algorithm as designed for single-cpu computation, but implemented the computation of objective values as sub-processes. During each generation, the overall population was divided into three sub-populations, and three PVM tasks were spawned to three different machines. These tasks each received copies of the chromosomes of one third of the population, evaluated them, and returned the results.

Because of the large populations involved in snake hunting (between 25,000 to 40,000 individual representations) we quickly exhausted the memory of the PVM machines. This resulted in a computational slowdown, and an occasional kernel shutdown. In an attempt to conserve memory, we have designed a

new representation which involves converting the chromosomes from an integer to a bitwise representation. Our calculations show that this will result in a significant memory savings.

However, this modification presents a new challenge as the PVM messaging scheme is not well suited to passing user-defined data types. In order to pass bitstring chromosomal representations between sub-populations, these strings will have to be translated or cast to some native C data type before transmission, and interpreted or re-cast upon receipt. These conversions will extract a one-time per message computational cost, and should not significantly affect effectiveness.

We are also working on a final implementation which will use a load-averaging scheme and the bitwise representation to evenly distribute memory usage and computation among the PVM machines. In previous implementations, the actual objective function processing was performed on separate machines using sub-populations: *copies* of approximately one third of the overall population. However, all other genetic operations, as well as the structures for the overall population (including representations, objective evaluation, and individual statistics) were performed and maintained on one machine. This creates a limiting factor equal to the memory capacity of the controlling machine.

In this final scheme, no machine is burdened with maintaining the overall population. Instead, the whole population resides in approximate thirds on each of the three machines, giving us a significant memory savings. There is still one machine that must act as the “master” or “parent” machine, and perform selection based on the fitnesses of individuals in the sub-populations. However, by using dynamic load averaging, it will be possible to decrease the sub-population size for this master machine so that it will not be overtaxed.

Computation of objective function values takes place on each of the three machines, and the results are placed in an “evaluation list.” These lists are sent to the master machine for participation in the GA selection procedure. The three evaluation lists are concatenated in a specific order so the results may be returned to the correct machine. Selection takes place on this list of objective values, and a list of mating pairs is generated. This list will consist of pairs of individuals, with each individual represented by its position in its respective sub-population.

This mating list is evaluated to determine the best distribution of population members for maintaining an even load on the PVM machines, redistribution lists are generated, and if necessary, individual population members are shifted from one population to another so that this load averaging can be achieved. For instance, assume that AIX1 is the master, and AIX2 and AIX3 are the remaining machines. Evaluation would take place as follows:

1. For each pair in the mating list, determine whether or not both individuals are on the same machine.
2. If the individuals are on different machines then:
 - (a) Place the least fit individual on the redistribution list of the higher fit individual.
3. Else if the individuals are on the same machine already then:
 - (a) Place both individuals on the redistribution list for their original machine.
4. When the complete mating list has been processed in this way:
 - (a) If necessary, reduce the population size on the redistribution list of the master machine proportional to its computational task and the overall population size.
 - (b) If necessary, shift individuals from the remaining redistribution lists until an even computational load is achieved for each of these machines.

Each machine then receives two lists. The first list contains the identification number of individuals who are going to be shifted to other populations, and which population they are going to. The second list is the mating pairs. Upon receipt of these two lists, each machine sends to the new populations the individuals (if any) that no longer belong in its population. Once a machine has received its mating list and new members (if any), and sent out its old members (if any), it proceeds with crossover and mutation according to the matches made in the mating list. The GA then proceeds in its normal manner.

7 Results

The GA has been quite effective in generating snakes in dimensions 7 and 8: it found the maximum length snake of 50 in dimension 7 and generated a one time record of 89 in dimension 8. Parallelization has been limited by the amount of machine resources available, specifically memory, but initial experiments have shown promise for significantly decreasing computational time.

8 Conclusions

We feel that the larger population sizes enabled by parallelization will give us much better results than the single-processor GA. The use of PVM as a parallelization platform gives us a flexible toolkit with which to implement these and other parallelization schemes. Future work will include using PVM's ability to present an abstract-machine software layer on non-homogeneous architectures to extend the PVM network beyond its current three machine cluster. We also plan on implementing fine and coarse grained parallel schemes in which each sub-population is isolated from the others and subjected to genetic operators separately. Best fit individuals are then allowed to slowly "bleed" over into the other populations.

References

- [1] Abbott, H.L., and M. Katchalski, On the Snake-in-the-Box Problem, *Combin. Theory*, Vol. 45, pp. 13-24, 1988.
- [2] Abbott, H.L., and M. Katchalski, Further Results on Snakes in Powers of Complete Graphs, *Discrete Mathematics*, Vol. 91, pp. 111-120, 1991.
- [3] Adelson, L.E., R. Alter, and T.B. Curtz, Long snakes and a characterization of maximal snakes on the d-cube, *Proceedings, 4th SouthEastern Conference on Combinatorics, Graph Theory and Computing, Congr. Numer.* 8, pp. 111-124, 1973.
- [4] Danzer, L., and V. Klee, Lengths of Snakes in Boxes, *Combinatorial Theory*, Vol. 2, pp. 258-265, 1967.
- [5] Davies, D.W., Longest -Separated- Paths and Loops in an N Cube, *IEEE Trans. Electronic Computers*, Vol. 14, p. 261, 1965.
- [6] Davis, L., (ed.) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York 1991.
- [7] Deimer, K., A New Upper Bound for the Length of Snakes, *Combinatorica*, Vol. 5(2), pp. 109-120, 1985.
- [8] Douglas, R.J., Upper Bounds on the lengths of circuits of even spread in the d-cube, *Combin. Theory*, Vol. 7, pp. 206-214, 1969.
- [9] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Co., 1989.
- [10] Harary, F., J. P. Hayes and H. J. Wu, A survey of the theory of hypercube graphs, *Comput. Math. Applic.*, 15 (1988) 277-289.
- [11] Holland, J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975.
- [12] Juric, M.L., Optimizing Genetic Algorithm parameters for Multiple Fault diagnosis applications, in *Proceedings of the IEEE Conference on Artificial Intelligence Applications (CAIA '94)*, February, 1994.

- [13] Kautz, W.H., Unit-Distance Error-Checking Codes, *IRE Trans. Electronic Computers*, Vol. 7, pp. 179-180, 1958.
- [14] Klee, V., A method for constructing circuit codes, *Assoc. Comput. Mach.*, Vol. 14, pp. 520-538, 1967.
- [15] Klee, V., What is the maximum length of a d-dimensional snake? *Amer. Math. Monthly*, Vol. 77, pp. 63-65, 1970.
- [16] Liepins, G.E., M.R. Hilliard, J. Richardson, and M. Palmer, Genetic Algorithm Applications to Set Covering and Traveling Salesman Problems, in *OR/AI: The Integration of Problem Solving Strategies*, (Brown, ed.), 1990.
- [17] Miller, J.A., W.D. Potter, R.V. Gandham, and C.N. Lapena, An Evaluation of Local Improvement Operators for Genetic Algorithms, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 5, pp. 1340-1351, Sept/Oct 1993.
- [18] Potter, W.D., B.E. Tonn, M.R. Hilliard, G.E. Liepins, S.L. Purucker, and R.T. Goeltz, Diagnosis, Parsimony, and Genetic Algorithms, in *Proceedings of the Third Int. Conf. Industrial & Engineering Applications of AI and Expert Systems*, pp. 1-8, July, 1990.
- [19] Potter, W.D., J.A. Miller, and O.R. Weyrich, A Comparison of Methods for Diagnostic Decision Making, in *Expert Systems with Applications: An International Journal*, Vol. 1, pp. 425-436, 1990.
- [20] Potter, W.D., J.A. Miller, B.E. Tonn, R.V. Gandham, and C.N. Lapena, Improving The Reliability of Heuristic Multiple Fault Diagnosis Via The EC-Based Genetic Algorithm, in *APPLIED INTELLIGENCE: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, Vol. 2, pp. 5-23, 1992.
- [21] Potter, W.D., R. Pitts, P. Gillis, J. Young, and J. Caramadre, IDA-NET: An Intelligent Decision Aid For Battlefield Communications Network Configuration, in *the Proceedings of the Eighth IEEE Conf. on Artificial Intelligence for Applications (CAIA '92)*, pp. 247-253, March, 1992.
- [22] Geist, Al, et.al, *PVM 3 Users's Guide and Reference Manual*, Oak Ridge National Laboratory, Oak Ridge, TN, 1993.
- [23] Singleton, R.C., Generalized Snake-in-the-Box Codes, *IEEE Trans. Electronic Computers*, Vol. 15, pp. 596-602, 1966.
- [24] Snevily, H.S., The Snake-in-the-Box Problem: A New Upper Bound, *Discrete Math* (to appear).
- [25] Starkweather, T., S. McDaniel, K. Mathias, D. Whitley, and C. Whitley, A Comparison of Genetic Sequencing Operators, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 69-76, 1991.
- [26] Tovey, C.A., *Polynomial Local Improvement Algorithms in Combinatorial Optimization*, Ph.D. Dissertation, Stanford University, Palo Alto, CA, 1981.
- [27] Wojciechowski, J., A new lower bound for Snake-in-the-Box codes, *Combinatorica*, Vol. 9, pp. 91-99, 1989.