Parallel Genetic Algorithm for Search and Constrained Multi-objective Optimization

Lucas A. Wilson, Michelle D. Moore, Jason P. Picarazzi, Simon D. San Miquel
Computing and Mathematical Sciences
(lwilson,mmoore,jpicarazzi,ssanmiguel)@sci.tamu.edu

Abstract

This paper introduces the design and complexity analysis of a parallel genetic algorithm to generate a “best” path for a robot arm to follow, given a starting position and a goal in three-dimensional space. Path generation takes into account any obstacles near the arm. This algorithm uses multiple optimization criteria, independent cross-pollinating populations, and handles multiple hard constraints. Individuals in the population consist of multiple chromosomes. The complexity of the algorithm is the number of generations processed times O(N^2) where N is the total number of individuals used for path generation on all of the optimizations. This research is being sponsored by NASA grant NAG 9-1401.

1. Introduction

NASA’s robot arm currently used on the space shuttle to manipulate various payloads is 15 meters in length with six different joints, or degrees of freedom. The robot arm on the International Space Station has seven degrees of freedom and is 17.6 meters in length. Each arm currently requires two astronauts to operate. One astronaut manipulates the arm via the control panel, and another monitors the arm’s progress. The “best path” for the arm to take to complete the task at hand is generated on the ground. The task of path generation involves rigorous calculations to find a reasonably short path for the arm to take. The arm must keep clear of any obstacles, including the shuttle itself. The long term goal of this project is to decrease the amount of human work involved in the path generation. This paper introduces a parallel genetic algorithm to be used to find a “best” path for the robot arm to follow. The starting position and goal are given in three-dimensional space, and the path generator takes into account any obstacles near the arm.

Genetic algorithms have been applied for some time to search and optimization [7] and to approximate solutions to NP-complete problems [5]. The use of a genetic algorithm to solve a problem involving multi-objective optimization is also not new. In fact, several multi-objective evolutionary algorithms exist. The Pareto Envelope-based Selection Algorithm (PESA) [2] is one such algorithm. In [1], the authors propose a steady state GA that handles multi-objective optimization. Another multi-objective optimization algorithm is the NSGA-II, described in [4]. The NSGA-II has been shown to have complexity O(mN^2). The algorithm proposed in this paper will be shown to have complexity O(N^2).

The remainder of the paper is organized as follows. Section 2 describes the problem, the goals of the algorithm and discusses the difficulty of the problem. Section 3 outlines the standard genetic algorithm and explains how and why this algorithm differs. Section 4 describes the types of processes and the flow of control within these processes. Section 5 details the design of and discusses the computation costs of each genetic operator. Section 6 presents the overall complexity of the algorithm. Section 7 reports our conclusion and suggests areas for future study.

2. The Problem

The difficulty of the problem stems from the numerous objectives that the trajectory generator must meet and from the multiple constraints that must be simultaneously satisfied. Each unique position of the robot arm may be represented by a listing of the absolute angle measurements of each degree of freedom on the arm; thus, each arm state is a point in seven-dimensional space. A trajectory for the robot arm will then consist of a sequence of these points. The problem is now to find one such trajectory that is most desirable, such as the shortest path that does not crash into an obstacle.

The Traveling Salesperson Problem (TSP) is often used as a basis of comparison for the effectiveness of various optimization techniques when applied to an intractable problem. The TSP requires a search of a finite search space in order to find a solution of a finite size. Determining a solution involves finding the permutation of locations that will result in the shortest total distance for the traveler. Finding the best solution in the case of the TSP would require the use of an algorithm of efficiency O(N!) since the solution space consists of a one dimensional list of size N!. The problem of path
generation for the robot arm is similar to the TSP in that the goal of both is to find a shortest path. The robot arm path generator needs to search a finite, eight-dimensional space bounded by the precision of the movement of the robot arm itself. An exhaustive search on the entire solution space for the robot arm trajectory problem would involve running the equivalence of a traveling salesperson algorithm for every possible combination of any number of points in the space. These results would then need to be compared to find the optimal path. Thus, it is easy to see that the problem of finding an optimal path for the robot arm is also intractable.

3. Terminology and System Configuration

3.1. The Genetic Metaphor

In 1975 John Holland introduced the idea of combining directed randomness with adaptation as a search and optimisation technique [9]. A genetic algorithm begins with a population of initial coded "guesses" (chromosomes) to a solution. An application specific fitness or objective function is applied to each chromosome in the individual, and based on the results, the better chromosomes are identified to "survive". Survivor chromosomes are paired (mating), then spliced together to form a new generation (crossover). Occasionally, portions of chromosomes are randomly altered (mutation). This process of selecting survivors based on fitness determination, mating, and mutation is repeated for a given number of generations, or until the chromosomes improve sufficiently to achieve a predefined goal. This approach has been widely applied to approximate solutions for NP-complete problems [5].

For this application, each individual has multiple chromosomes. Mating occurs between individuals and crossover occurs between the chromosomes of the mated individuals. Multiple populations of individuals are used to facilitate optimization on different criteria. Periodically individuals from a population are copied and other populations are “pollinated” with these values. Standard mutation of genes creates divergence and increases the search space by providing new alleles. In order to perform “mutation” of the eighth dimension (the path length), two additional operators are used Davidor [3]. Individuals are initialized with a random number of chromosomes. Addition duplicates a randomly chosen chromosome and inserts the duplicate next to itself. Deletion picks a chromosome at random and deletes it.

3.2. Hardware Configuration

This algorithm has been tailored for execution on Beowulf clusters. The GA processes, each consisting of a spatially isolated population working on a different objective, are distributed across the cluster nodes. This provides asynchronous processing capability to minimize idle processor time on all nodes. The gigabit Ethernet connection provides for efficient pollination of the individual process.

3.3. Individual Encoding

In order to encode a complete path in each individual, a variable-length string approach is used. Each individual consists of multiple chromosomes and summary information about the individual (Fig. 1). This summary information includes a real value to represent fitness, an integer value for the number of chromosomes in the individual, and a pointer to the array of chromosomes. Each chromosome represents a state of the robot arm. An individual’s collection of chromosomes represents the potential path (or trajectory) that will be evolved in an attempt to find an optimized path. Each chromosome contains seven genes. Genes are represented as real values where each value corresponds to one of the seven degrees of freedom in the robot arm. The gene values represent shoulder pitch, shoulder yaw, shoulder roll, elbow pitch, wrist pitch, wrist yaw, and wrist roll for the arm position.

![Figure 1: Encoding Scheme for Individual](image)

3.4. System Architecture

The algorithm uses a master/slave topology (Fig. 2). The master acts as the beehive, a process dedicated to pollination. Each slave process handles a different search or optimization objective such as power consumption, running time, or generation of applicable paths (discovery). After pollination of individuals with different optimization criteria, good characteristics from one individual will be sent to other objective optimization processes, thereby creating individuals that have been optimized for all of the objectives. Each slave communicates with the master only when pollination needs to occur.
4. Process Types and Control Flow

4.1. Master Process (Beehive)

The beehive is a process dedicated to controlling pollination. The beehive contains a collection of individuals from various search and optimization objectives collected during pollination. Each processor receives a partition of the beehive of size $P$, the pollination rate of that particular processor. Once the beehive has collected individuals from a processor, it leaves behind copies of $P_a$ individuals, chosen probabilistically, from the partitions of the other processors. By altering the partitions within the beehive, more important objectives’ individuals will be distributed in greater quantities to a greater number of processes.

4.2. Objective Processes

Processes not devoted to searching for valid paths will be used to optimize paths on various functions. These may include, but are not limited to the minimization of number of arm states, the minimization of total execution time, the minimization of power consumption, and the minimization of total arm motion.

Many constraints will have to be recognized to generate valid and efficient paths. These include ensuring that the arm’s joints do not exceed specified tolerance limits, ensuring that no other objects collide with the arm while it is in motion, and ensuring that the arm’s end effector is properly aligned with the target. Whenever new gene values (alleles) are generated, they are bounded by their corresponding joint tolerance limits. Detecting collisions and proper end effector alignment will be performed in simulations used to determine individual fitness for the search objective.
beehive only as necessary. The flow (Fig. 3) of the discovery process closely models a traditional generational GA, with the addition of pollination and of addition and deletion, another form of mutation. Optimization processes follow the same pattern, but do not have addition and deletion operations.

5. Genetic Operator Design and Analysis

5.1. Initial Population

To generate the initial population, each individual is given a random length. Its chromosomes are then filled with random values for the genes. The range for each of the random values is limited to be within the physical constraints of the robot joints.

for each individual in a population
    generate random number of chromosomes
    for each chromosome in individual
        for 1 to 7 degrees of freedom
            generate random value

The computation cost for initialization is \( N \times \#\text{chromosomes} \times 7 \).

5.2. Calculating Fitness

Determining individual fitness is different for each separate objective. For simple optimization objectives, such as minimization of number of chromosomes, the computation cost is \( N \). With more complex optimization objectives, such as minimization of arm motion, a chromosome by chromosome traversal of each individual becomes necessary, giving a computation cost of \( N \times \#\text{chromosomes} \).

Calculating fitness for the search objective is more difficult to analyze. Each individual must be tested through simulation, in order to ensure proper final alignment and to ensure that the path is free of collisions with other objects. Therefore computation time for calculation of fitness for the search objective will be \( t_4 \times N \), where \( t_4 \) is the average time of simulation for an individual. Although constant, \( t_4 \) will be very large relative to the time of a single computation on the system, it will likely remain the dominating factor.

Because of this time requirement for fitness calculation, each individual’s fitness will be saved. Fitness is calculated only once per individual when that individual is created, after it has moved due to pollination, or after it is mutated.

5.3 Performing Selection

In order to improve execution time, the selection algorithm is a modification on tournament selection. By holding \( d \) tournaments, each with \( N/d \) competitors, and keeping the best of each tournament, selection can be performed in a computation time of \( N/d \), with \( d \) survivors per generation.

for 1 to d
    select N/d competitors
    choose most desired

5.4. Pollination

Pollination occurs immediately following selection. Each processor reports its readiness to pollinate with the beehive. Upon establishing communication, \( P \) individuals are copied to the beehive. Then \( P_a \) individuals are received from the beehive, all of which originally belonged to other processes. \( P \) and \( P_a \) are the pollination rate and the acceptance rate of a processor, respectively. Upon reception of new individuals, their fitness is recalculated for the processor’s objective. Each step in pollination, outgoing and incoming, has a computation time of \( 2 \times N \), not accounting for communications time.

Pollination is performed after selection to ensure that optimizations made on other processors are carried over into the next generation of individuals on the current processor. If selection were performed after pollination, there would be no guarantee that individuals added through pollination would survive to perform crossover.

signal readiness to pollinate
for 1 to P
    send individual to holding area
for 1 to P_a
    receive individual from holding area

5.5. Mating

Selecting parent pairs for crossover will be done by randomly choosing two individuals from the survivors of selection and the new individuals from pollination. By using random selection instead of a selection method based on fitness, individuals introduced through pollination are more likely to be selected for crossover, allowing for the introduction of new building blocks [7] from other populations. Random selection of parent pairs can be done with a computation time of \( N \).

5.6. Crossover

As is the case with fitness, crossover techniques will vary depending on a processor’s objective. For the search
objective, a modified Analogous Crossover [3] is used because of its minimal path disruption (Fig. 4). In order to find a suitable cross point in the second parent, each of the second parent’s chromosomes is compared for total variance in arm status from the randomly chosen cross point in the first parent. The variance is calculated by summing the absolute differences in each corresponding gene. The selected cross point will be the chromosome in the second parent with the smallest change in the arm’s state.

Once a suitable cross point is located in the second parent, crossover occurs by copying the appropriate chromosomes from each parent into a new individual (Fig. 5). Traversal of the second parent in order to locate a suitable cross site and creation of a new individual are both performed in a computation time of \( N \). Crossover for an entire population therefore requires a computation time of performed in \( 3 \times N^2 \).

\[
\text{for each pair of parents } (P_1, P_2) \\
\text{choose a random chromosome } A \text{ in } P_1 \\
\text{for each chromosome } B \text{ in } P_2 \\
\sum_{g=1}^{7} \left| B_g - A_g \right| \\
\text{choose } B \text{ in } P_2 \text{ with smallest variance from } A \\
\text{for each chromosome } \alpha \text{ in } P_1 \text{ on the interval } [0, A] \\
\text{add } \alpha \text{ to new individual } C \\
\text{for each chromosome } \beta \text{ in } P_2 \text{ following } B \\
\text{add } \beta \text{ to new individual } C
\]

For optimization objectives uniform crossover, a more disruptive crossover technique, is used to help maintain diversity within the population [6] [8]. The genes of each chromosome in a pair of parents are crossed in order to produce two new individuals. Because traversal of the chromosomes is necessary for each pair of parents, the time complexity for a single crossover operation has a computational cost of \( N \) and of \( N^2 \) for an entire population.

5.7 Addition and Deletion

To prevent loss of diversity in string lengths, the deletion and duplication addition operators proposed by [3] are used to provide background noise over the string lengths (Fig. 6). Each of these operations requires traversal over each of the individuals it is performed on, so the computational cost for both operations on an entire population is \( 2 \times N^2 \).
5.8 Mutation

The proposed mutation operation calls for the modification of a single gene within a single chromosome of an individual. Modification of the gene will be performed by replacement with a random value. No traversal of individual chromosomes is necessary if using a direct-addressing implementation for chromosome strings. Because of this, the computational cost for mutation is $O(N)$.

for each new individual created with crossover
if mutation should occur
    pick a random chromosome C
    pick a random gene g
    replace $C_g$ with a random value

6. Overall Complexity

Each chromosome requires initialization individually with time complexity for an individual $O(N)$, and for the population, $O(N^2)$. Determining the fitness of an individual when the objective function is a complex optimization requires a chromosome by chromosome traversal of each individual. This has a time complexity of $O(N^2)$. Holding $d$ tournaments, each with $N/d$ competitors allows selection to be performed in $O(N)$ time. Each step in pollination, both outgoing and incoming, is performed in $O(N)$. Crossover for a population is performed in $O(N^2)$. Time complexity for addition and deletion is $O(N^2)$. Time complexity for mutation is $O(N)$.

Each of these operations is performed asynchronously on each processor, in order to ensure maximum use of processor time. The combined computational cost for each generation is:

$$N \times \text{ #chromosomes} + N/d + 4 \times N + 6 \times N^2.$$

This is an overall time complexity of $O(N^2)$.

7. Conclusion and Future Work

This paper details a novel design for a Parallel Genetic Algorithm for Search and Constrained Multi-objective Optimization. By distributing objectives, through different fitness functions, across spatially-isolated populations and independently controlling pollination rates for those populations, optimization on multiple objectives and discovery of valid paths in an eight-dimensional, bounded search space can be accomplished with a time complexity of $O(N^2)$, regardless of the number of objectives.

Implementation of this design is ongoing. Currently this algorithm is designed for discovery of paths for a seven degree-of-freedom, fixed shoulder robot arm. Future work will include expanding the algorithm for a non-fixed shoulder to be more applicable to the Space Station robot arm. Other future work will be focused on experimenting with different crossover and mutation techniques, in order to determine the most efficient techniques to use for this particular application.

Acknowledgements

William A. Jackson and Juan S. Villarreal have contributed and continue to contribute to this project. This research is being sponsored by NASA grant NAG 9-1401.

References


