Genetic Algorithms, their Applications and Models in Nonlinear Systems Identification.

Frank Lup Ki Wan

B.A.Sc. University of British Columbia, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

March 1991

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of E E

The University of British Columbia
Vancouver, Canada

Date Apr 16 91

Abstract

The Genetic Algorithm was used to estimate the hydraulic compliance of the hydraulic system on the UBC teleoperated heavy duty excavator. Using real recorded and simulation data from the excavator, the Genetic Algorithm has successfully identified the compliance of single link and multi-link hydraulic system of the excavator.

A Parallel GA (PGA) was implemented with 16 T800 Transputers. It achieved a speedup factor of 12 over a traditional GA. With such a high speedup factor, real-time monitoring of hydraulic compliance and other hydraulic parameters is becoming possible.

New mechanisms such as the distributed fitness function, the active error analysis were used to enhance the performance of a PGA. A PGA which incorporated these mechanisms actually outperformed a traditional GA in key areas such as variance of the estimated parameter and parameter tracking ability.

Finally, a physical model that explains the fundamental properties of GAs was introduced. The physical model (a hypercube) not only provides an excellent explanation of GAs searching power, but also gives insight to GAs users ways to improve and to predict the performance of GAs in most applications.

Table of Contents

List of Figures

Acknowledgements

I am grateful to my supervisors Dr Lawrence and Dr. Dumont for being so patient with me, to let me explore many different interesting areas, and to see me through writing my thesis.

I like to thank K. Kristinsson for permitting me to use the core of his Genetic Algorithm program and his help on translating his program from Pascal to C. I am especially grateful to N. Sepehri for his help and the use of his detailed documentation on the hydraulic system of the UBC teleoperated excavator.

I will always remember and grateful for the encouragement and support from Darrell Wong, Dan Chan, Real Frenette, Edward Lam and David Warkintin.

Finally, I am indebted to my family. Their faithful support has made this work possible.

Chapter 1
Introduction

1.1 What are Genetic Algorithms ?

Genetic Algorithms (GAs) are search procedures based on the mechanics of natural genetics and natural selection. They extract the ideas of survival of the fittest from nature to form a robust search mechanism. GAs are a parallel, global search technique that can simultaneously search through many points in the parameter space. GAs differ from other search techniques by the use of concepts taken from natural genetics and evolution theory.

1. GAs use finite length codings of the parameters (usually in binary), instead of the parameters themselves (The binary string coding of parameters is necessary for the Crossover and Mutation process).
2. GAs work with a population-by-population approach rather than point-by-point techniques.
3. GAs need only fitness values (dependent on application). There is no requirement for derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules instead of deterministic ones.

A simple GA consists of three operators: Reproduction, Crossover and Mutation. Reproduction is the survival of the fittest within a GA. Based on the survival of the fittest principle, it decides which strings are going to survive and which ones are going to disappear according to their normalized fitness values. Individuals with above average fitness will have more offsprings than those with below average fitness. This step directs the search towards the best individuals. Crossover is a randomized yet structured operator that takes valuable information from both strings (parents) and combines it to find a highly fit individual. To apply this operator, 2 individuals are mated at random and a point on the interval $1 < k < N-1$ (N = length of the binary encoded strings) is chosen randomly. Two new strings are then created by exchanging all characters between positions 1 and k inclusively. For example, given the following two binary encoded strings:

$$A_0 = \text{'0000000'} \qquad B_0 = \text{'1111111'} \tag{1.1}$$

and with crossover point at 4, the new strings will be:

$$A_1 = \text{'1111000'} \qquad B_1 = \text{'0000111'} \tag{1.2}$$

The Reproduction and Crossover operators are responsible for most of the searching power of the Genetic Algorithm. The Mutation operator is used to introduce new information into the population at the bit level. It acts as an insurance policy against the loss of important genetic material at a particular position. Figure 1.1 shows a simplified block diagram of a genetic algorithm. A more rigorous explanation of how the GA work is in Holland's "theory of schemata"[11].

The GA always converges toward the best from the algorithm's point of view. However there is no guarantee that it will converge to the optimum (the true value). Nevertheless it has many advantages over other "identification methods"

Identification methods such as recursive least squares, instrumental variable method are based on the principle that a smooth search space exists. These recursive schemes are in essence local search techniques that search for zero gradient by going in a direction suggested by the local gradient. Thus they often fail in the search for a global maximum if the search space is not differentiable and linear in the parameter space. Statistical techniques will do better in multimodal functions (which have multiple optima), but they require too much computation time for higher dimensional problems. GAs have been shown to behave well on multimodal functions. They work much more subtly than just a random search with preservation of the best. Their ability to handle functions that are nonlinear and discontinuous makes them good candidates in function maximization and system identification.

GAs do require massive computation power since they are population rather than single point based algorithms. However, GAs are inherently parallel. All individuals in a population evolve simultaneously without central coordination. Many parallel implementations of the GA [18][17][16][24][25] for function maximization have been demonstrated recently. With the advent of parallel processing, GAs are becoming a practical on-line identification method for many "difficult" problems.

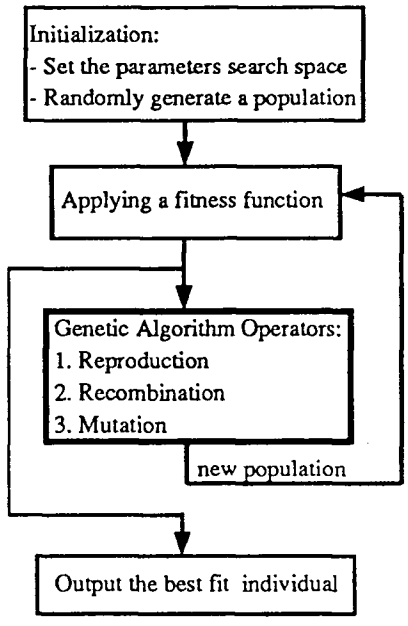


Figure 1.1: A typical Genetic Algorithm flow chart

To demonstrate how a GA works, a second-order system of the following form is used:

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_0 \times (u(k) + b_1 \times u(k-1)) \tag{1.3}$$

Where a_1, a_2, b_0, b_1 are the unknown parameters. Assuming the range of the unkown parameters are,

$$-5 < a_1 < 5; \quad -5 < a_2 < 5; \quad -5 < b_0 < 5; \quad -5 < b_1 < 5 \tag{1.4}$$

the unknown parameters are converted into binary numbers. The number of bits (N) for each parameter depends on the required resolution. N can be calculated with the following equation:

$$\begin{aligned} resolution &= \frac{(upper\ bound - lower\ bound)}{2^N} \\ or: \quad N &= log_2\left(\frac{(upper\ bound - lower\ bound)}{resolution}\right) \end{aligned} \tag{1.5}$$

The binary numbers are concatenated to form an individual string as:

$$B(a_1) \mid B(a_2) \mid B(b_0) \mid B(b_1) \tag{1.6}$$

If a resolution of 0.01 is used for all the parameters, such an individual string length will be 40 bits long.

The choice of the population size depends strongly on the individual string length [8]. For string length of 40, a population size between 100 to 400 should be used[8]. A random number generator is initially used to generate all individuals in the population.

For discrete parameter identification, a fitness function can be defined as:

$$fitness = \sum_{w=0}^{window\ size} (Bias - (y(k-w) - \hat{y}(k-w))) \tag{1.7}$$

where:

y(k) = measured output value of the system.

$\hat{y}(k)$ = calculated output value based on the estimated parameters.

Bias : a positive constant that ensures positive fitness for good estimated parameters.

Window : used to minimize the estimated parameters' variance.

Refering to Figure 1.1, after fitness values of all individuals in the population are calculated, the GA operators such as Reproduction, Crossover and Mutation will operate on the corresponding individual binary strings. The binary string format allows GA operators to create a new generation of binary strings. The search for a better individual (parameters) will continue as shown in Figure 1.1

1.2 Scope of the thesis

Das and Goldberg[5] have shown that a GA can be used in estimating discrete-time parameters. They have demonstrated the feasibility of a GA in identifying parameters a second-order system and suggested future GA extensions to more complex problems.

Kristinsson and Dumont [13] have made use of the unique properties of GAs to directly estimate poles and zeros of a system. By doing so, the design of an adaptive pole-placement controller for such system is simplified. They found that:

1. A GA convergence rate is similar to that of Recursive Least Square (RLS) in terms of number of input-output data points.
2. A GA gives unbiased estimates in the presence of colored noise.

3. A GA could be used in cases where the system is not linear in the unknown parameters (RLS cannot be applied).

Because of its unique advantages, GAs can be applied to areas where conventional identification methods perform poorly or fail. A good example of such an area is the hydraulic system in a typical heavy duty mobile machine, e.g. an excavator, where it is very difficult to apply conventional identification methods. For example, the compliances (defined later) of the hydraulic system are very important since they affect the controllability of the hydraulic system.

In order to apply GAs in real-time system identification, it is necessary to run the GA on multiple processors. There are several ways to parallelize GAs. It is important to determine the achievable speedup factor, efficiency of a parallel implemented GAs (PGAs) that are used for system identification.

Many papers have been published on using the PGA in function maximization. However, system identification requirements are quite different from that of function maximization. The performance of PGAs used for system identification can be further enhanced or optimized by various techniques. Thus it is possible that in certain applications, PGAs can outperform traditional GAs in key areas such as convergence rate of estimated parameter and variance of the estimated parameters.

This thesis is organized into the following chapters:

1. Chapter 2 describes the operation of the teleoperated heavy duty hydraulic excavator and the related backgrounds of the excavator hydraulic system. It also illustrates mechanisms that can cause the compliance of the system to vary.
2. Chapter 3 shows how a GA is used to estimate single link and multiple links hydraulic compliances on the excavator. It also presents one possible way to estimate mult-link hydraulic compliance on-line with GAs.
3. Chapter 4 discusses ways that GAs can be parallelized. It suggests ways to improve the speedup factor and efficiency of the parallel architectures for PGAs. A PGA is implemented to estimate the hydraulic compliance in the excavator hydraulic system (Chapter 3). The performance of such PGA and traditional GAs is compared.

4. Chapter 5 discusses ways to improve PGAs performance in system identification. Various techniques will be used and their effectiveness on PGAs performance will be examined.
5. Chapter 6 offers physical insights on the searching power of GAs.
6. Chapter 7 will summarize all the above ideas and suggests future research.

Chapter 2

Related background

2.1 Teleoperated heavy duty hydraulic machine

Figure 2.1 shows a typical manual joint mode control found on excavator machines today. The control consists of two two-degree-of-freedom joysticks with a one-to-one mapping between the joystick motions and the links. This method of control is very easy to implement but requires extensive training time for most operators to master. In the Telerobotics Lab at UBC Electrical Engineering, resolved motion control is used on the prototype teleoperated heavy duty hydraulic excavator. Resolved motion control, which is based on end point trajectory technique [23], provides a much more "natural" motion interface to the operators.

There are many benefits of applying teleoperation and computer control to heavy duty hydraulic machines:

1. Computers can create a better human interface, such as resolved motion control, to the machine. While it may take one year for operators to be proficient in the manual joint mode control, most operators can usually master the resolved motion control in a much shorter period. Resolved motion can also increase the operator productivity.
2. Advanced control algorithms provide a more efficient and safer way to control the hydraulic actuators in the machine [23].
3. Overload sensing and stability checking can be added to a teleoperated machine. Such features can prevent dangerous manoeuvres by the operators. The safety of the operators and the machines are thus increased.

Figure 2.1 shows the physical block diagram of the hydraulic system of a conventional manually operated CAT 215B excavator. The major components of the system are:

1. **Pilot valves**: manual control for the pilot pressure which determines the spools displacement in the main hydraulic valves. The displacement of the spools in the main valves determine the orifices areas of the main values.

2. **Main valves**: It is the heart of the hydraulic system. It consists of 2 independent branches. Each branch is connected to a constant flow pump. Branch #1 contains the bucket and the boom valves and branch #2 contains the swing and the stick valves. A cross-over network consists of pressure operated valves which connects both branches together. The cross-over network allows either pump to provide surplus power from one branch to other branch when needed. The schematic of the main valves is in Appendix A.
3. **Hydraulic actuators**: The boom, the stick and the bucket are moved by linear hydraulic actuators. The swing is powered by a hydraulic motor. All the input and output lines of the hydraulic actuators and motor are connected to the main valves by flexible hydraulic hoses.

For teleoperated control, the conventional pilot valves are replaced with a new set of servo-valves. The servo-valves will respond to the control voltage and build up the corresponding pressure to move the main hydraulic spool to the desired displacement. Pressure sensors are installed on the input and output lines of the hydraulic actuators (swing motor, boom, stick, bucket). Resolvers are installed on the mechanical joints to measure joints angles. The block diagram of the refitted hydraulic system is illustrated in Figure 3.1.

2.2 The need for an on-line diagnostic program

Heavy duty hydraulic machines such as excavators are designed to operate in a rugged environments. Periodic maintenance will help cut down the repair frequency of machines. Yet for hydraulic machines operating in a rugged environments, component breakdown is very common. Many machine failures are not foreseeable and repairs are done on as-required basis. In forestry, major repairs are difficult for machines operating in remote areas. The downtime of such machines can be as high as 30% of the machine's life time.

Some experienced operators can readily sense malfunction or performance degradation of the hydraulic machines. With the manufacturers' diagnostic procedures and the operators' experience, faulty or damaged components can be isolated and the necessary repairs made.

The control of a teleoperated heavy duty hydraulic machine is similar to the "fly-by-wire" control on advanced aircrafts such as Airbus 300 series. The operator's joystick motion is translated into

commands by the computer and the corresponding hydraulic actuators motions are executed using some control algorithm.

In a closed loop control system, the operator is isolated from the physical machine. Using the prototype teleoperated excavator as an example, if there was a leakage in the boom hydraulic actuator , both experienced and inexperienced operators could not observe any steady-state error in boom motion since the closed-loop control would have compensated for the leakage. If one of the hydraulic hoses was worn out or damaged, the operator may detect some degradation in the machine performance, but he would rely on the computer to perform the proper diagnosis. For reliable and safe operation of any teleoperated hydraulic machine, the onboard computer must periodically perform on-line self-checking, which involves identifying some important hydraulic parameters of the hydraulic circuit.

With the help of a proper on-line diagnostic program, it may be possible to identify early signs of most common mechanical problems, such as worn out hydraulic hoses, dirt build up in the hydraulic valve, and leakage in hydraulic actuator, before they become major problems later. The on-line diagnostic program will reduce the downtime of the machine.

2.3 How the compliance of the hydraulic system changes

The hydraulic compliance of a typical mobile hydraulic machine, i.e. an excavator, used in the primary industry can be defined as:

$$C_{system} = C_{fluid} + C_{structure} \tag{2.1}$$

$$C_{fluid} = \frac{V_{fluid}}{\beta_{fluid}} \tag{2.2}$$

where

C_{system} = the compliance of the hydraulic system

C_{fluid} = the compliance of hydraulic fluid

$C_{structure}$ = the compliance of the flexible hose and the surrounding structure.

V_{fluid} = volume of the fluid

β_{fluid} = the bulk modulus of hydraulic fluid

2.3.1 Hydraulic fluid (C_{fluid})

The compliance of the hydraulic fluid changes with time due to aeration. Air can be dissolved or entrained in hydraulic fluid. However, the bulk modulus of the hydraulic fluid, β is affected by undissolved free air in the fluid (Figure 2.3, reprinted from [19]).

Under high pressure, more air can be dissolved in hydraulic fluid. However as air bubbles are compressed and passed through the pump, they do not dissolve immediately. Figure 2.4 (reprinted from [19]) shows the rate of solution of air bubbles in hydraulic oil. Very often tiny bubbles of air accumulate on the surface of the hydraulic actuator or valve, causing the hydraulic compliance of the actuator to increase by an order of magnitude[19].

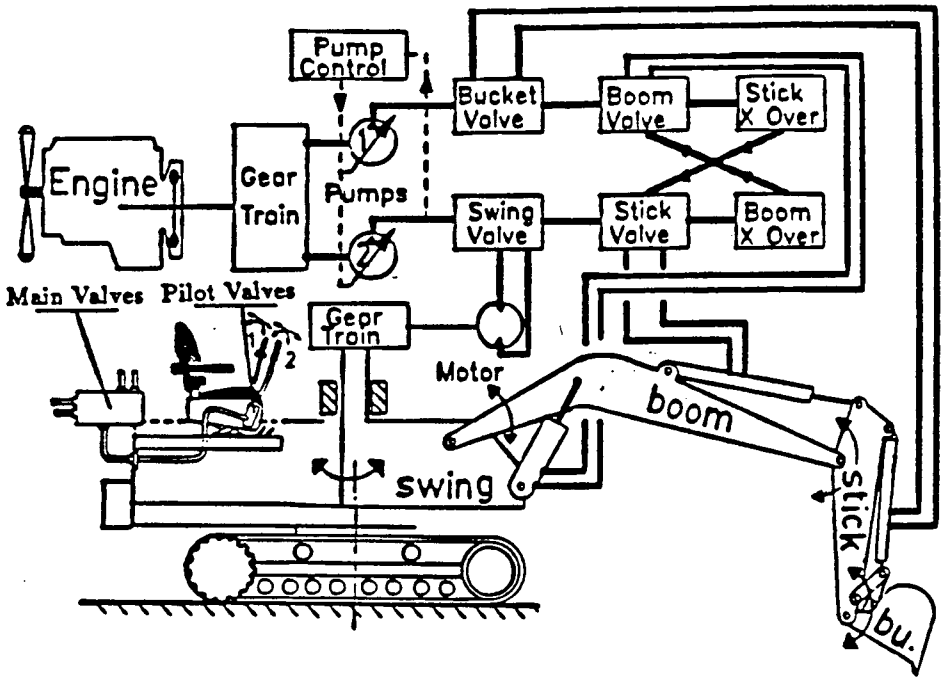


Figure 2.1: Caterpillar 215B excavator

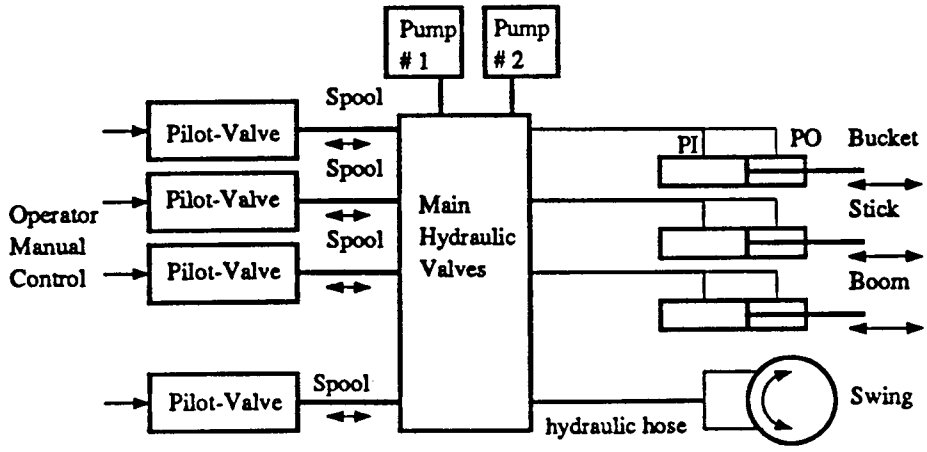


Figure 2.2: The physical model of a conventional hydraulic system on a CAT215B excavator

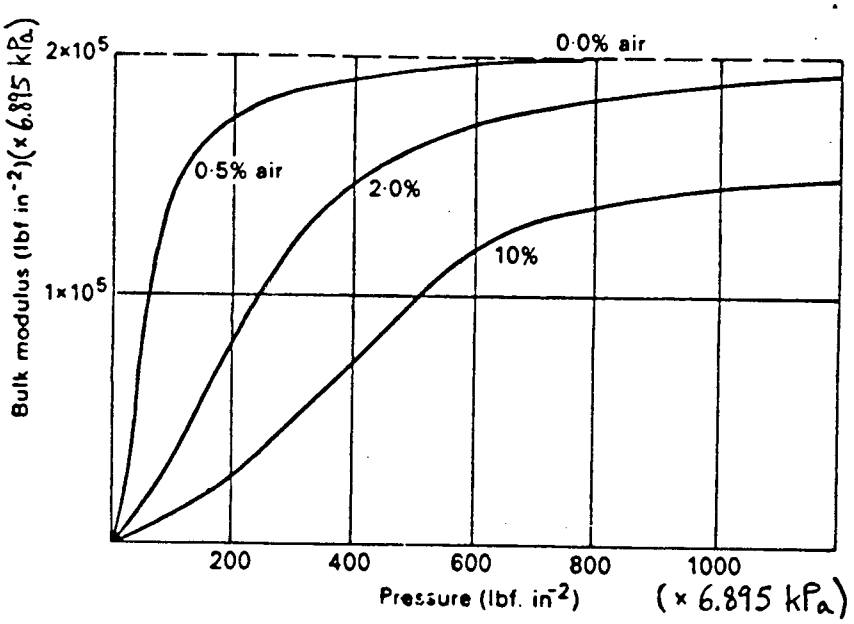


Figure 2.3 Effect of pressure on the bulk modulus of an air-oil mixture. The air is free.

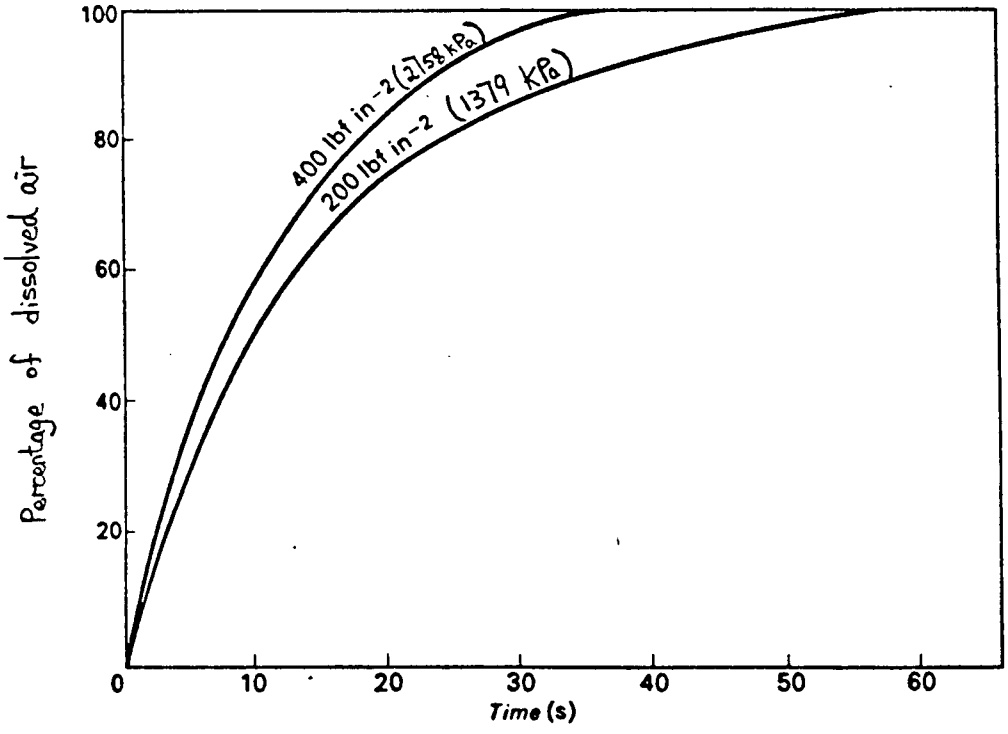


Figure 2.4 Rate of solution of air bubbles in hydraulic oil

2.3.2 Mechanical problems ($C_{structure}$)

There are many other conditions such as worn out hydraulic hose, leakage inside the hydraulic actuator and seal failure that could cause the compliance of the system to deviate.

Leakage in actuator or seal failure

In a manually operated hydraulic machine, if there is a significant leakage (i.e. 5% to 10%) in the hydraulic actuator, the operator will notice some loss of power and unwanted drift motion in the associated link. In a teleoperated hydraulic machine, the closed-loop control would compensate for the loss of power and drift motion. Thus the closed-loop control could mask the symptoms of the leakage or improper sealing in the hydraulic actuator from the operator. If unattended, this minor problem may evolve into a major machine failure and the safety of the machine operator will be compromised.

Referring to the boom hydraulic circuit in Figure 2.5, if there is no leakage in the actuator:

$$Q_i = C_i\dot{P}_i + A_i\dot{X} \tag{2.3}$$

where :

Q_i = *flow at the pump side*

C_i = *the hydraulic compliance*

$\dot{P}_i$ = *change in pump side pressure*

A_i = *area of the pump side cylinder*

$\dot{X}$ = *velocity of the cylinder rod*

Assume there is a leakage in the actuator, Q_l flows from input chamber along the wall of the cylinder to the output chamber.

$$Q_i' = C_i'\dot{P}_i' + A_i\dot{X} + Q_l \tag{2.5}$$

where :

$Q_i' = new\ flow\ at\ the\ pump\ side$

$C_i' = apparent\ hydraulic\ compliance$

$\dot{P}_i' = new\ change\ in\ pump\ side\ pressure$

$Q_l = leakage\ in\ the\ boom\ actuator$

Because of the closed-loop control, the error in boom angular velocity is small in steady state and $\dot{X}$ remains unchanged. However the transient response of the system is quite different and the abnormality can be detected by monitoring the compliance.

Combining the above two equations,

$$\begin{aligned} Q_l &= C_i' \dot{P}_i' - C_i \dot{P}_i \\ &\leq \left(C_i' - C_i\right) \dot{P}_i \qquad since\ \dot{P}_i \geq \dot{P}_i' \end{aligned} \tag{2.7}$$

Thus the change in compliance will be at least proportional to the leakage:

$$\left(C_i' - C_i\right) = \Delta C_i \geq \frac{Q_l}{\dot{P}_i} \tag{2.8}$$

If the compliance of the hydraulic circuit is monitored periodically, then the abnormal deviation in the measured compliance would be detected and warning can be sent to the controller.

Voltage-spool transfer function drift

Drift in voltage-spool transfer function can be caused by a worn out spool or poor pressure regulation in the servo-valves. Since such problems affect the flow directly, the measured compliances are affected as well.

Referring to the boom hydraulic circuit in Figure 2.5, the sensitivity of steady state flow with respect to pressure and spool displacement can be seen in Table 2.6 which is derived from the hydraulic equations for boom (Appendix A). For example, an error of 0.02 inch (8 %) in spool displacement causes a 50% difference in flow Q_{BOi}.

In the prototype teleoperated excavator designed at the UBC Robotic Lab, an advanced control algorithm based on a computed flow control is used [23]. This control algorithm consists of feedforward compensation and conventional proportional-derivative feedback. This feedforward compensation technique decouples the nonlinearities in the hydraulic and joint structure of the excavator. With most of the nonlinearities compensated by the feedforward term, the controller can use smaller proportional Kp and derivative Kv gain in the feedback to correct for minor unmodelled dynamics. The controller will be more stable and performs much better than a simple P-D feedback controller.

The feedforward term involves calculating the desired spool displacement based on the line pressure readings and the desired joints velocities. Using the spool-voltage transfer function (the inverse of voltage-spool transfer function, Appendix B), the control voltages for the servo-valves can be found. Thus any offset drift in the servo-valve voltage-spool transfer function will cause the feedforward term to be incorrect. From Table 2.6, an error as high as 50 percent in the feedforward term is possible. The controller performance will deteriorate significantly since the proportional and derivative feedback have to compensate for all 50% of the highly nonlinear flow error.

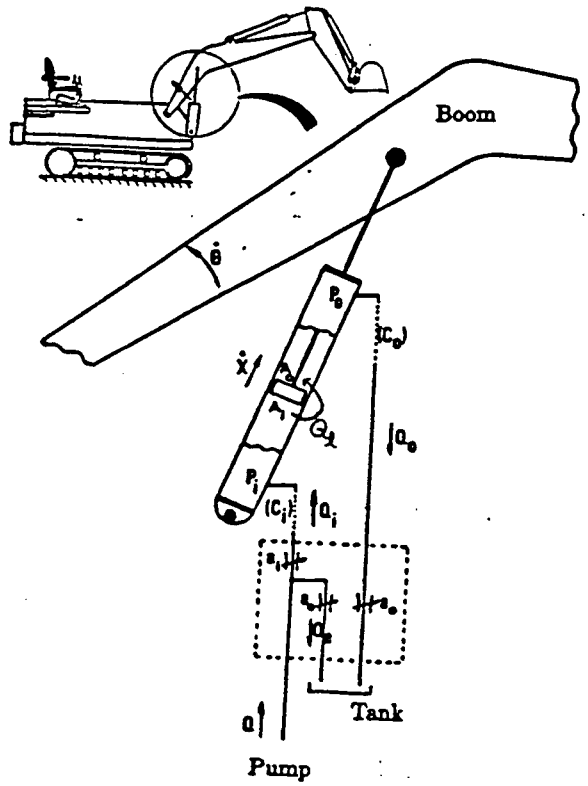


Figure 2.5: Schematic of a heavy-duty hydraulic actuated boom

	P_{BOi}@ [2030 psi](13996 kPa)	P_{BOi} @ [1830 psi](12617 kPa)	P_{BOi}@ [1630 psi](11239 kPa)
Q_{BOi} [in^3/s](cm^3/s)	Spool [in](cm)	Spool [in](cm)	Spool [in](cm)
[60] (983.4)	[0.2741] (0.696)	[0.2688] (0.6827)	[0.2627] (0.6673)
[50] (819.5)	[0.2650] (0.673)	[0.2592] (0.6584)	[0.2527] (0.6419)
[40] (655.6)	[0.2561] (0.650)	[0.2499] (0.6347)	[0.2427] (0.6165)

Figure 2.6: Steady State Flow Q_{BOi} versus Pressure P_{BOi} and Spool Displacement

2.3.3 How hydraulic compliance affects the stability of a P-D controller

The effect of hydraulic compliance on the system under P-D closed-loop control was illustrated in [14]. Figure 2.7 shows a block diagram of a hydraulically actuated robot with P-D feedback control. The linearized spool and hydraulic actuator model are combined with the proportional and derivative feedback control to form an overall transfer function:

$$\frac{\Theta_i(s)}{\Theta_i^d(s)} = \frac{K_p a_0}{s^3 + a_2 s^2 + (a_1 + a_0 K_v)s + K_p a_0}$$

$$a_0 = \frac{2K_{xi} D_m}{J_i C_i}; \quad a_1 = \frac{K_{pi}}{C_i}; \quad a_2 = \frac{2D_m^2}{J_i C_i}; \tag{2.9}$$

Where:

Θ_i is the angle of the robot arm,

Θ_i^d is the desired angle of the robot arm,

K_p is the proportional feedback gain,

K_v is the velocity feedback gain,

C_i is the hydraulic compliance.

From the above equations, one can see that C_i determines the transfer function pole locations. Thus the stability and closed-loop performance of the hydraulic system depend heavily on the value

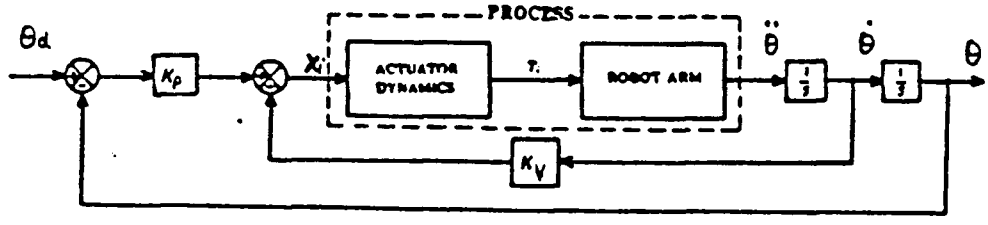


Figure 2.7: General block diagram of a hydraulically actuated robot with P-D feedback control

of C_i. Figure 2.8 (reprinted from [14]) shows the response of a P-D controlled hydraulic manipulator to a step input. In Figure 2.9 (reprinted from [14]) , the same K_p and K_v were used but with C_i increased by a factor of 4. The increase in C_i caused the response to be oscillatory.

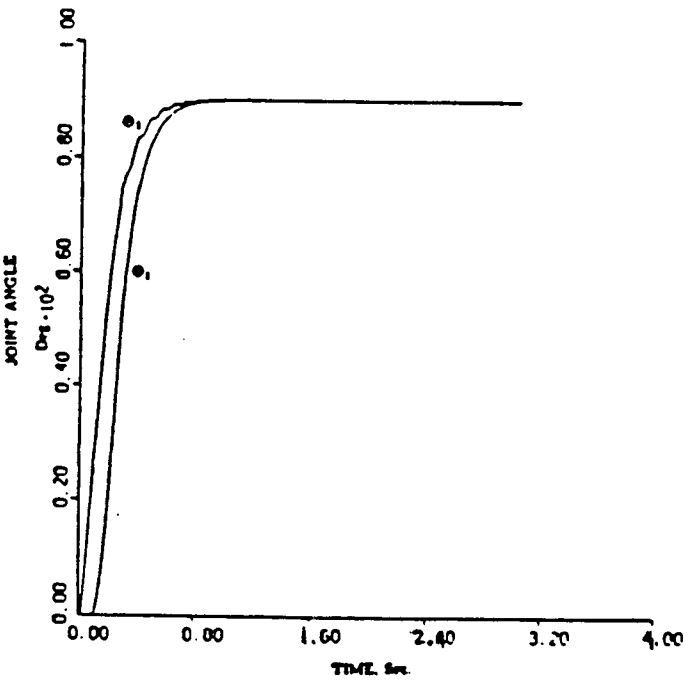


Figure 2.8 Respond of a P-D controlled hydraulic manipulator to step input

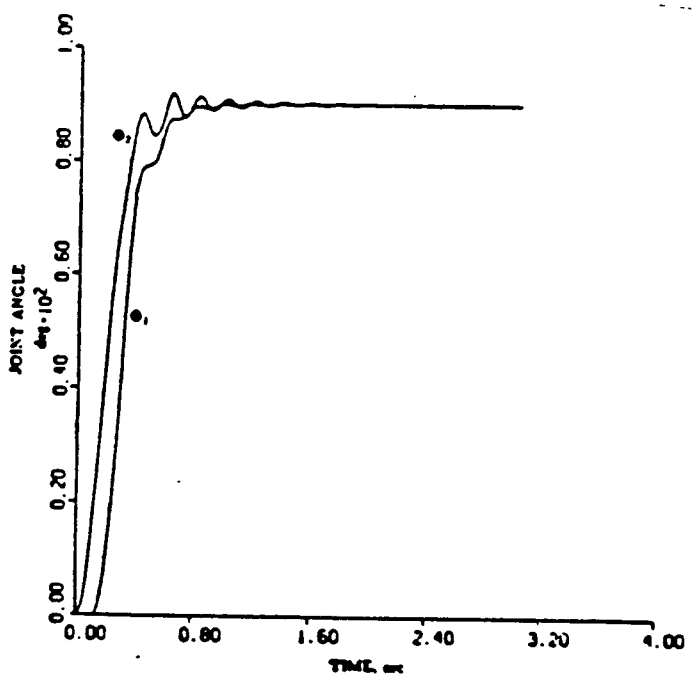


Figure 2.9 Response of a P-D controlled hydraulic manipulator with the decreased stiffness

2.3.4 Comments

In a teleoperated hydraulic machine, the performance of the controller is affected by the hydraulic compliance of the system. Since many other common problems such as worn-out hoses, trapped air bubbles, leakage in the actuator, and defective seals all affect the compliance, monitoring the compliance will give the diagnostic program an early warning from problems in the hydraulic system. With additional information and help from some other techniques, it is possible for the diagnostic program to isolate the causes. Proper action can be taken before the teleoperated heavy duty hydraulic machine performance starts to degrade significantly or major machine failures happen.

2.4 Difficulties in identifying the compliance of the hydraulic system

Difficulties

The main hydraulic valve is the heart of the hydraulic system. The schematic diagram of the hydraulic main valves is shown in Figure 2.10.

1. The complexity of the circuit is increased by the presence of the cross-over network. Depending on the internal pressures P_{12}, P_{13}, P_{22}, P_{23}, the hydraulic circuit can have 5 possible configurations (Figure 2.11).
2. The equations that govern the hydraulic system are nonlinear and the steady state solution can only be obtained by iteration method. Fig. 2.12 shows the configuration a, branch #1 and the set of equations that describe the hydraulic circuit. Note that the internal states P_{11}, P_{12}, Q_{11}, Q_{12} have to be solved by iteration method.
3. The hydraulic compliance can only be identified during the transient stage of the hydraulic actuator's motion, i.e. when the line pressures P_{BUi} and P_{BOi} are still rising. The number of data points available is small. The identification algorithm must have a fast parameter convergence rate.

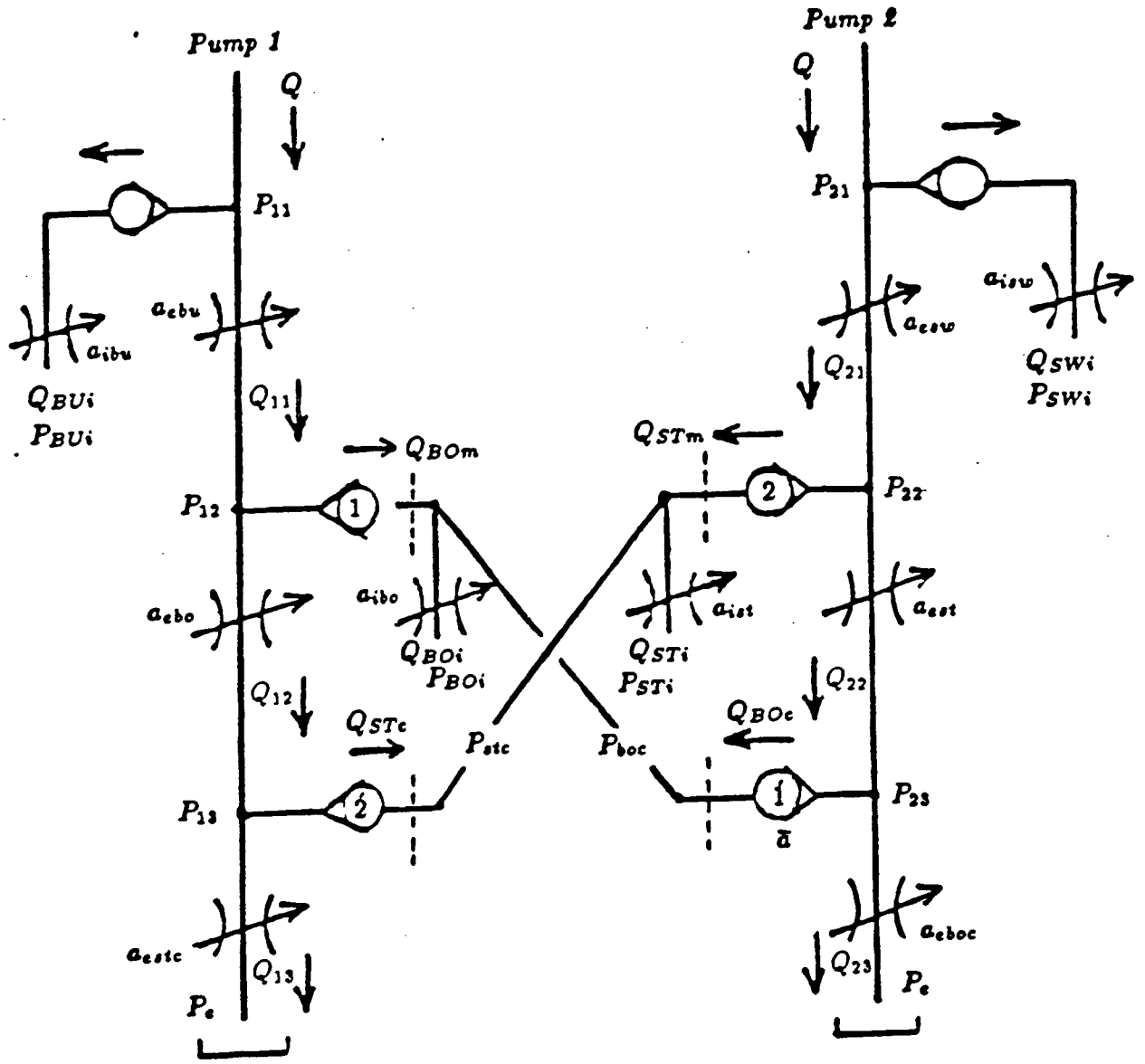


Figure 2.10: Schematic diagram of the hydraulic main circuit

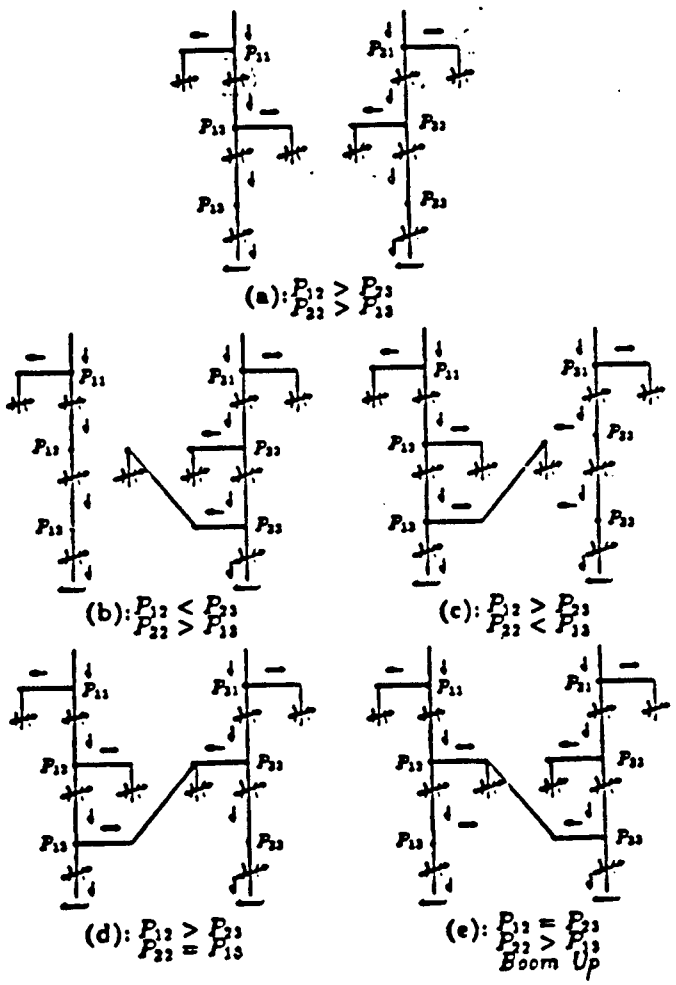


Figure 2.11 Alternative solutions to hydraulic main circuit

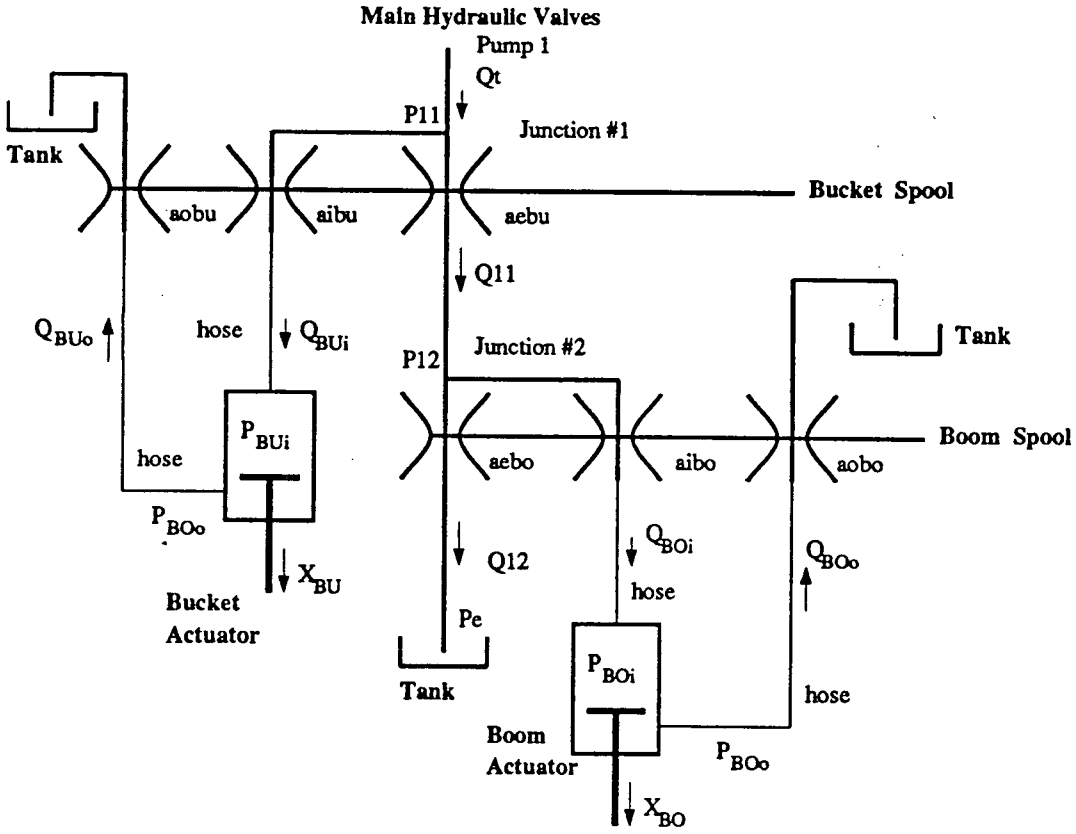


Figure 2.12: The branch #1 of the main hydraulic valves in configuration (a)

Chapter 3

Identification of hydraulic compliance with GA

3.1 Identify the compliance of a mobile hydraulic machine

The GA was used to identify the hydraulic compliance of the refitted hydraulic system of the prototype teleoperated CAT 215B excavator. The block diagram of the hydraulic system is showed in Figure 3.1.

Why the Genetic Algorithm ?

The Genetic Algorithm (GA) is chosen to identify the compliances of the system because:

1. The number of unknown parameters is small. There are 4 independent links and thus only 4 compliances C_{SWi}, C_{BOi}, C_{STi}, and C_{BUi} that need to be identified. The size of the global population for GA depends strongly on the number of unknown parameters. A large global population will require extensive computation power.
2. The unique feature of GA is that it does not put strict requirements on the model. Its population based searching algorithm can easily handle discontinuities and nonlinearities in a model. The GA does not require the hydraulic equations to be arranged in any form. There is no need to rearrange the equations to find a closed form solution for the internal states P_{11}, P_{12}, ... etc. .
3. Due to the cross-over network, the model of the hydraulic system is changing from time to time depending on the operator command and the external loading. It is very difficult to find an identification method, especially a recursive one that can handle such model changes. The GA is a nonrecursive algorithm. It stores the estimated parameters in its population, which is independent from changes in the model of the hydraulic system.
4. Previous work[5][12] showed that the GA parameter convergence rate is similar to (or slightly faster than) Recursive Least Square methods. Thus GA parameter convergence rate is sufficiently high for compliance identification.
5. Even though GA is computation intensive, it is a naturally parallelisable algorithm [17][24]. The GA speed can be improved with parallel processing.

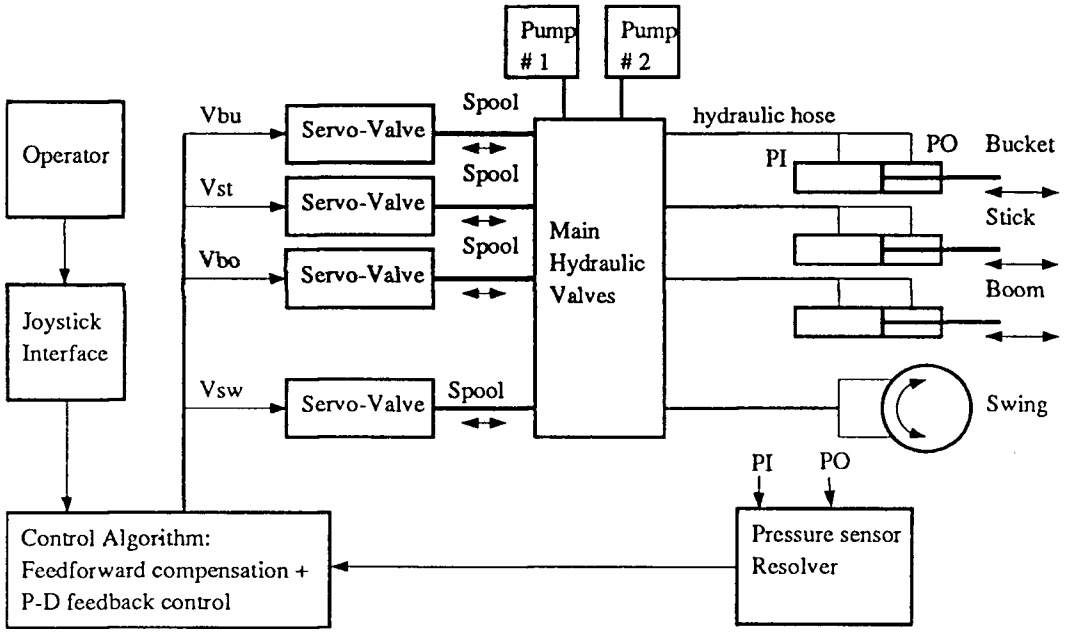


Figure 3.1: The block diagram of the refitted hydraulic system for teleoperation control

Other method (Recursive Least Square method)

Recursive Least Square (RLS) method can be applied to single link hydraulic compliance identification by linearizing the simple single link hydraulic equations. However,there are quite a few difficulties in applying RLS in multi-link example. First, because there is no closed form solution for P_{11} and P_{12} (internal states of the hydraulic system), it is difficult to formulate the constraints into the RLS formula. Second, since the hydraulic equations are highly nonlinear, there is no guarantee that the algorithm RLS will converge at all.

Although RLS is much less computational intensive (30 — 50 times less) than the conventional GA, it can not be used in the hydraulic area because of the problem nature.

3.1.1 Single link identification

The boom link was chosen to test GA performance in identifying the hydraulic compliance. Figure 3.2 shows a schematic of a heavy-duty hydraulically actuated boom of an excavator. The three orifices (ai, ae, ao) belong to the boom main valve and are controlled by the boom spool. The main valve is connected to the actuator through two flexible hoses (compliances Ci and Co). The side of the actuator which is connected to the pump with output flow Q is called the pump

side and the side which is connected to the tank is called the drain side. The pump pressure P is measured but it is too noisy to be useful. Line pressures Pi and Po are measured by pressure sensors and they are related to the flows as follows

Pump Side:

$$Q_i = A_i\dot{X} + \dot{P}_i C_i = ka_i\sqrt{P - P_i} \tag{3.1}$$

$$Q_e = ka_e\sqrt{P - P_e} \tag{3.2}$$

$$Q = Q_i + Q_e \tag{3.3}$$

Drain Side:

$$Q_o = A_o\dot{X} - \dot{P}_o C_o = ka_{eo}\sqrt{P_o - P_e} \tag{3.4}$$

where k is the discharge coefficient [20]. Pe is the tank pressure, Qe is the flow to the tank, and Q is the total pump flow (a known constant).

The above equations are used to identify parameters C_i and C_o. For example to identify C_i, the above equations are used as follows:

1- Given the measured values of Pi(m), where (m) denotes measured data, and the boom joint angle at time t, the first order derivative of the pressure $\dot{P}_i$ and boom angular velocity are determined by linear piece-wise least-square curve-fitting technique. ($\dot{X}(m)$ is proportional to the boom angular velocity)

2- A population consisting of binary encoded string $C_i(j)$, $_{(j=1,2,..n)}$ with its range defined already ($0.1 > C_i > 0.0$) is initially generated randomly. The above equations are used recursively as follows:

$$\hat{Q}_i = A_i\dot{X}(m) + \dot{P}_i(m)C_i(j) \tag{3.5}$$

$$\hat{P} = \left(\frac{\hat{Q}_i}{ka_i}\right)^2 + P_i(m) \tag{3.6}$$

$$\hat{Q}_e = ka_e\sqrt{\hat{P} - P_e} \tag{3.7}$$

$$Error = E(j) = \left| \; Q - \left|\hat{Q}_i\right| - \left|\hat{Q}_e\right| \; \right| \tag{3.8}$$

where $\hat{Q}_i$, $\hat{Q}_e$, $\hat{P}$ are the intermediate state information. Note that the measured pump pressure P is too noisy, the estimated one $\hat{P}$ is used in the calculation instead. The orifice areas are derived from the spool or the servo-valve voltage measurement (Appendix A). Absolute operators are used to prevent Error cancellation due to sign difference in the calculation

3- Step two is performed over a window size to minimize the estimated parameter C_i variance and the effect of noise. The fitness value of a particular $C_i(j)$ is defined as:

$$\begin{aligned} &fitness(C_i(j)) = \sum_{t}^{window} (B - E^2(j)) \leq \sum_{t}^{window} B \\ &where: \\ &window = 10 \\ &B = 400 \end{aligned} \tag{3.9}$$

The value of B (Bias) is chosen so that if $E^2(j)$ is larger than Bias, the fitness value of such $C_i(j)$ will become negative.

4- Based on the fitness value, $C_i(j)$ with high fitness value will be allowed to have more offspring. $C_i(j)$ with small or negative fitness value will be discarded from the population. GA operators (Crossover and mutation) will then generate a new population of $C_i(j)_{j=1,2,...,n}$. Step one to three are then repeated until the transient period is over (Pi(m) is steady). $C_i(j)$ with the highest fitness value will be outputed or recorded.

3.1.2 Implementation details

The recorded data was obtained from Real Frenette and Nariman Serpehri's feedforward compensation, open loop experiment on the prototype teleoperated CAT215B excavator. Figure 3.3a-d showed the recorded data of boom actuator line pressures (P_{BOi} and P_{BOo}) , servo-valve voltage and spool measurement, and boom angle versus time for a boom command angular velocity of 4 deg/s .

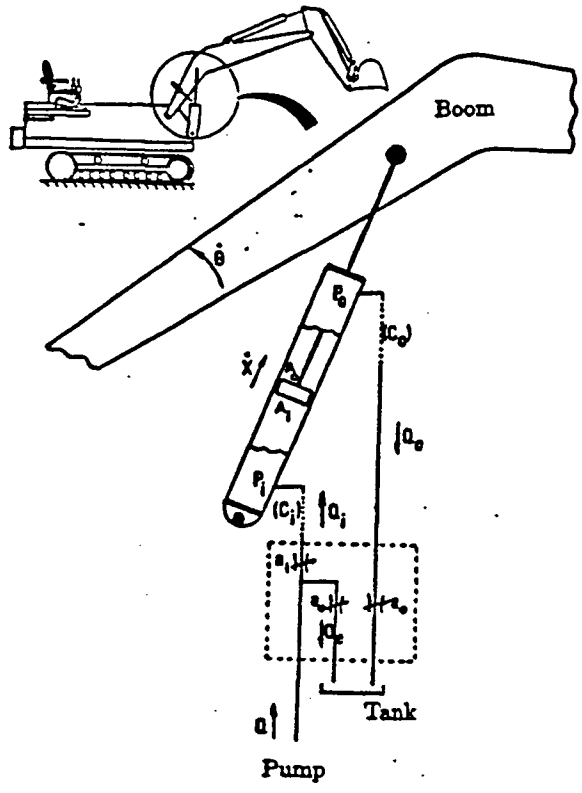


Figure 3.2 Schematic of a heavy-duty hydraulic actuated boom

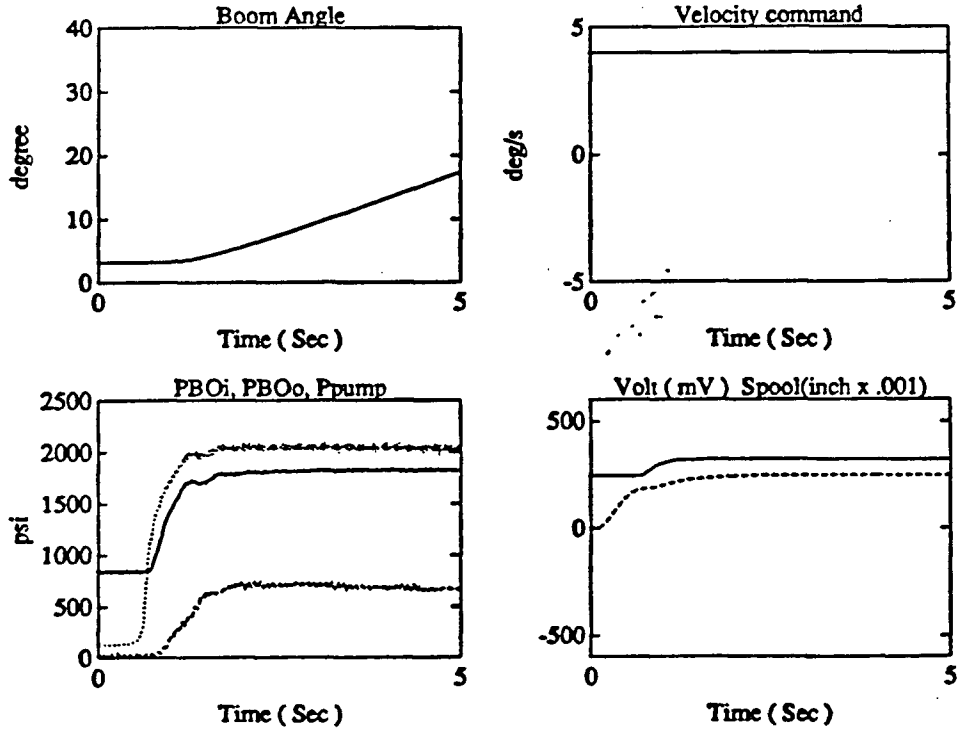


Figure 3.3 Data record

Preparation and Considerations

The accuracy of the mathematical model: The mathematical model should be a good approximation of the physical process. Otherwise, it would be difficult for GA or any other identification method to work properly. In the hydraulic model, the first-order pipe model is only an approximation of the transmission line dynamic. The model ignores the turbulence and all other second-order effects.

Filtering out noise in the measured data: Since GA is trying to identify hydraulic parameters during the transient period, it cannot perform much smoothing on the measured data (even if it is noisy) , because such an operation would eliminate the transient information. For example, the boom angle must be differentiated to obtain the boom angular velocity. If too many data points were used to find a smooth velocity profile, then the transient information of the boom angular velocity would be lost. The amount of smoothing on measured data should be kept to a minimum to preserve the transient information.

Figure 3.4 shows how GA converts the measured data into the variables used in the equations. The first order derivative $\dot{P}_{BOi}$ of the pressure and boom velocity are determined by a linear piecewise least-squares curve-fitting technique with number of data points limited to 5 samples.

The complete detail of the boom hydraulic system is listed in Appendix B.

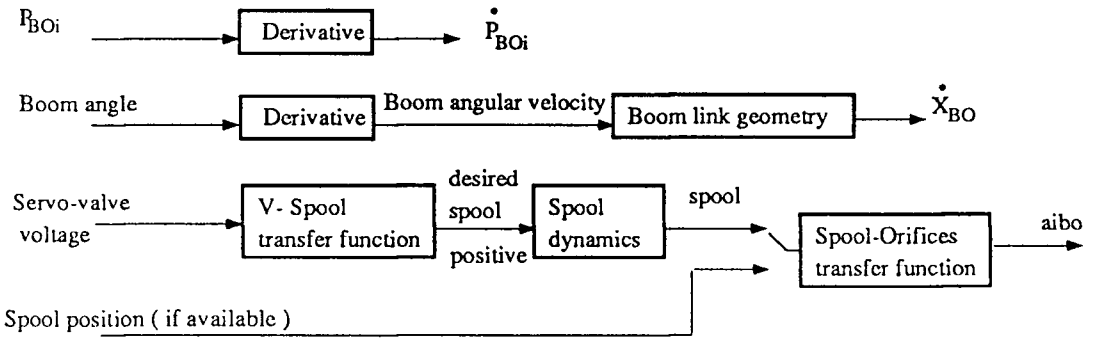


Figure 3.4: GA interface to the measurement data

3.1.3 Experimental Results

The first attempt to identify C_i failed. A negative value of C_i was obtained from the measured data. After comparing the boom, the spool, voltage measurement with the results from simulation,

it was confirmed that both the spool measurement and the voltage-spool transfer function had the same offset error. That both the voltage-spool transfer function and spool measurement had the same offset error may be due to the fact that the voltage-spool transfer function was originally derived from the same measuring equipment. An offset of 0.01 inch in spool measurement caused a steady state flow error of 20 percent.

In the following experiments, both spool and voltage measurement were compensated by the calculated offset.

Two experiments were performed. In the experiment #1, the spool measurement was used to calculate the orifice area (ai, ae, ao). In the experiment #2, the servo-valve voltage is used. If the servo-valve voltage is used (Figure 3.4), then the spool dynamics will be involved in orifice area calculation. Even though the spool dynamics is not known precisely and may affect the accuracy of the calculated orifice areas, the servo-valve voltage is preferred because direct spool measurement may not always be available (special equipment is needed).

The results of experiment #1 and #2 are in Figure 3.5 and 3.6 respectively. Line pressures P_{BOi}, P_{BOo} and boom motion are showed in Figure 3.5 a-b. Identification was performed during the pressure rise where the compliance C_{BOi} (C_i) was effective. The estimated C_{BOi} and Error (defined earlier) of both experiments are shown in Figure 3.5c-d and Figure 3.6c-d. In both experiments, during the period when GA was active, the Error was decreasing monotonically. The difference between estimated C_{BOi} from both experiments was small (0.005 versus 0.0086).

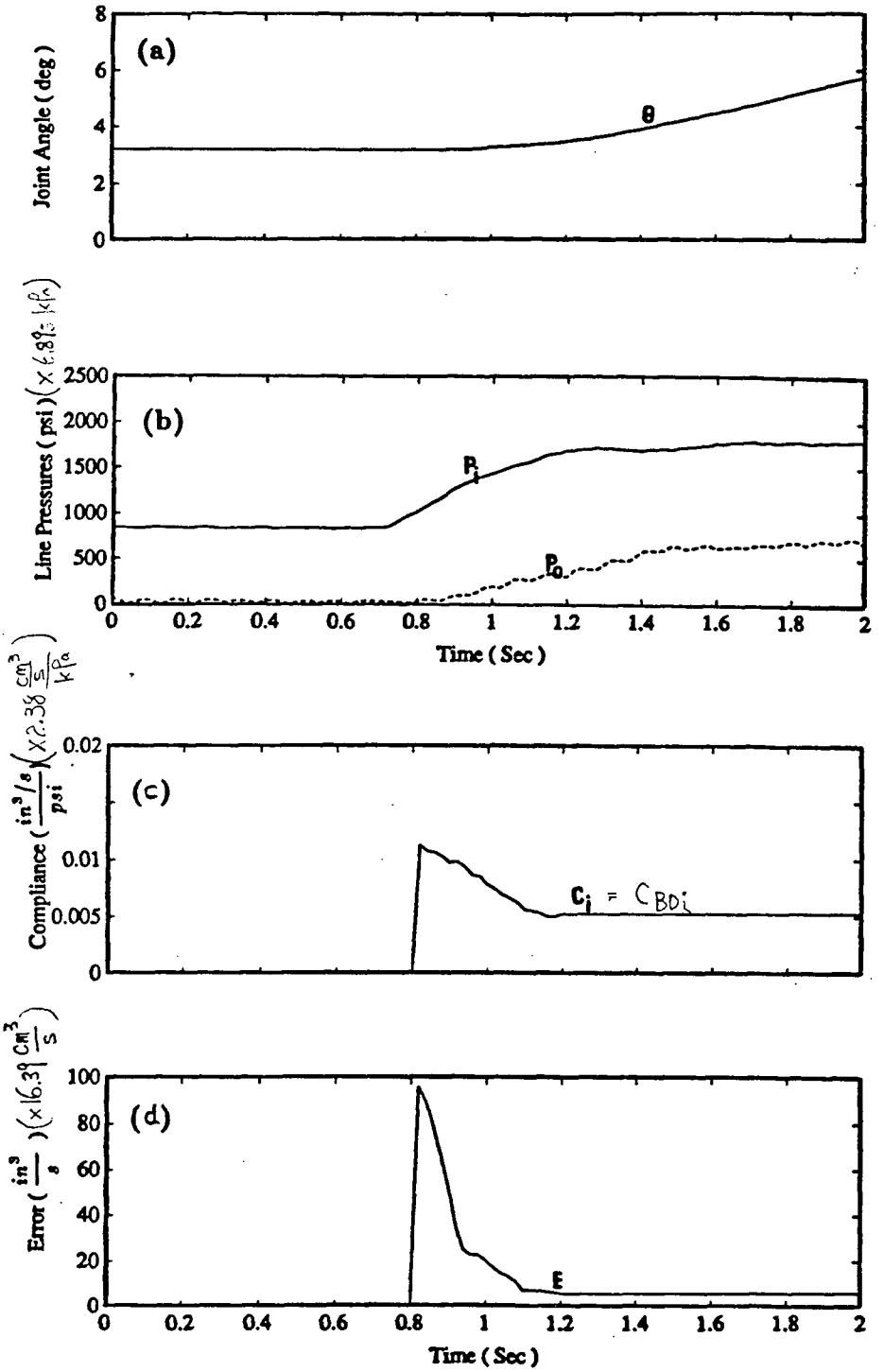


Figure 3.5 Experiment #1 (with spool data)

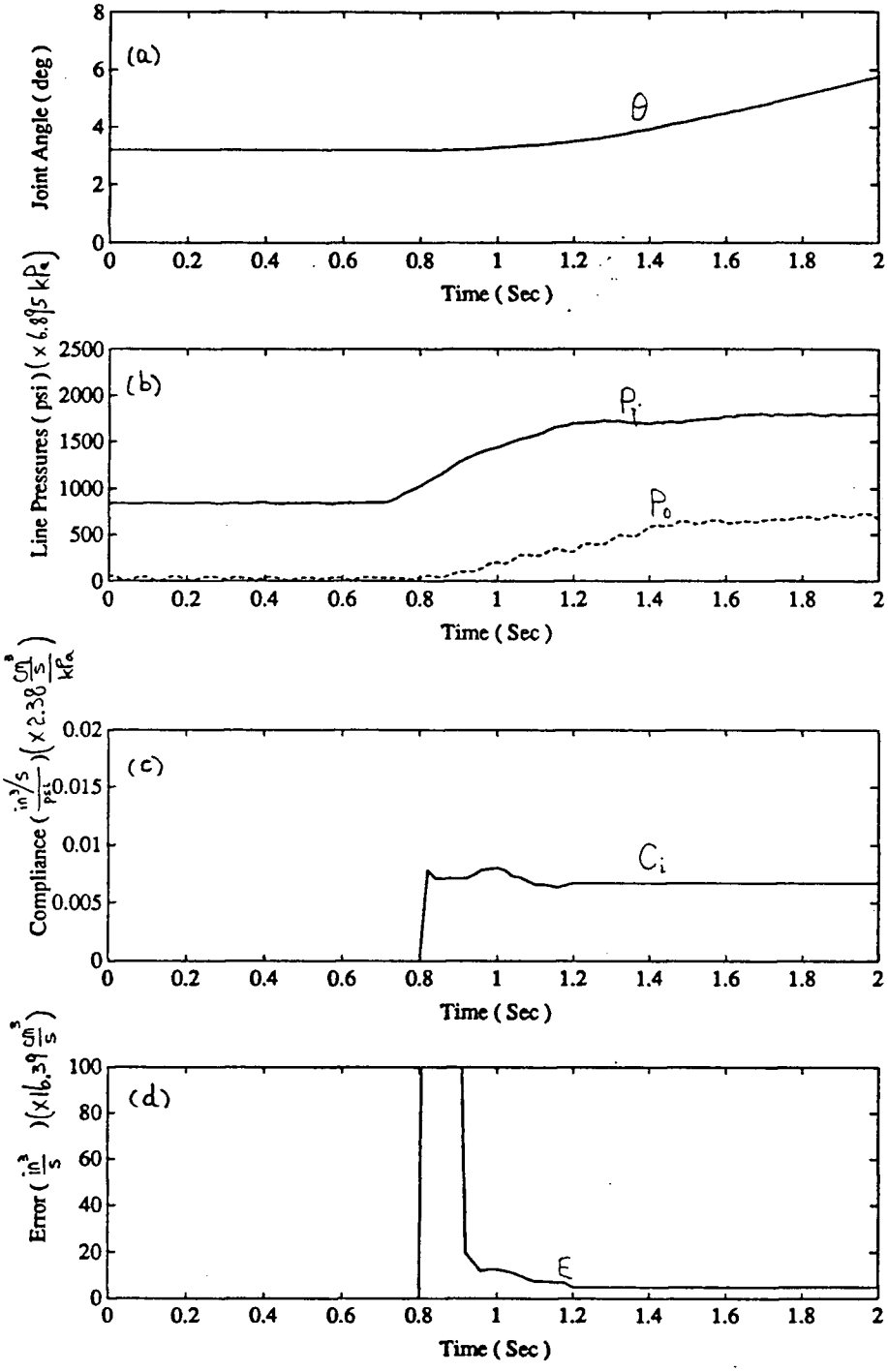


Figure 3.6 Experiment #2 (with servo-valve voltage data)

In order to test the accuracy of the estimated hydraulic compliance Ci, this estimated Ci value, with a different set of measured data (voltage, boom angle) are used in a simulation which would calculate the boom actuator input line pressure P_{BOi}. The calculated P_{BOi} would be compared with the measured P_{BOi}. Figure 3.7 shows the block diagram of the excavator model, GA and the simulator.

The line pressure P_{BOi} from the simulator and the measured P_{BOi} are plotted in Figure 3.8a. The agreement between both measured and simulated P_{BOi} was excellent, an indication that the estimated C_{BOi} (0.0086) was accurate. In order to see how C_{BOi} affect P_{BOi}, two different values C_{BOi} (0.0043 and 0.0172) were used in the simulator and the estimated P_{BOi} are plotted side by side with the measured P_{BOi} in Figure 3.8b. There was a significant difference between these three sets of curves. The two experiments with the simulator confirmed the accuracy of the estimated C_{BOi}.

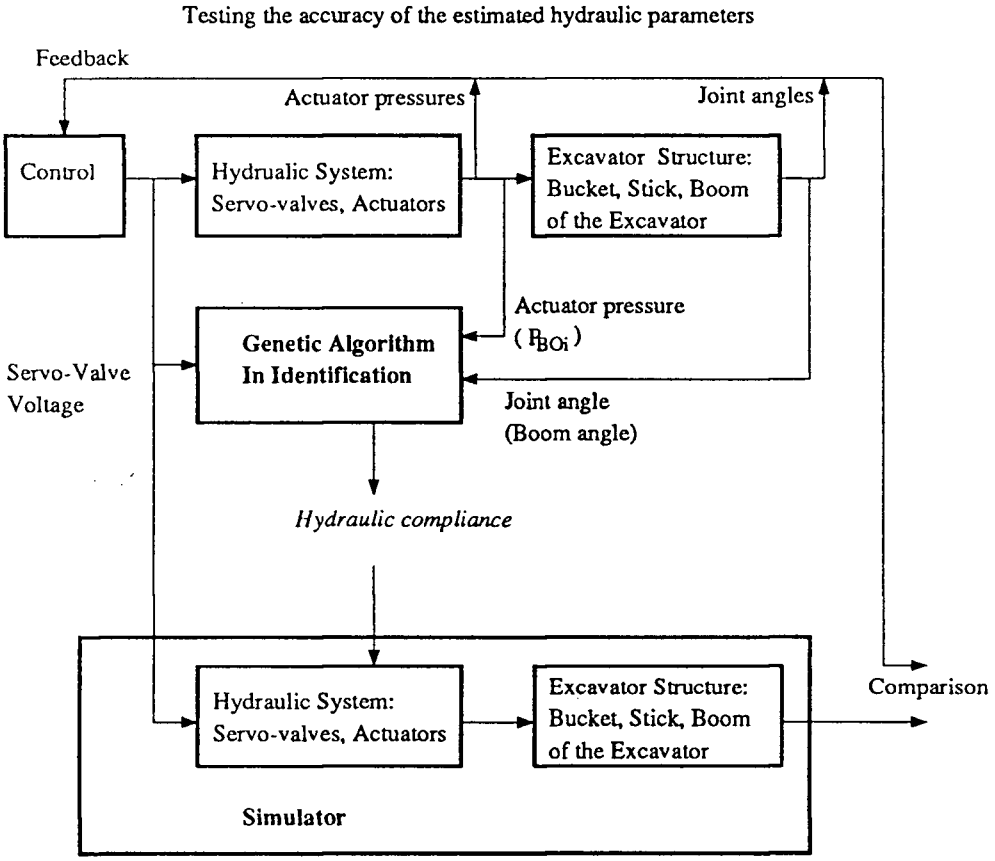


Figure 3.7: Block diagram of the identification and simulation setup

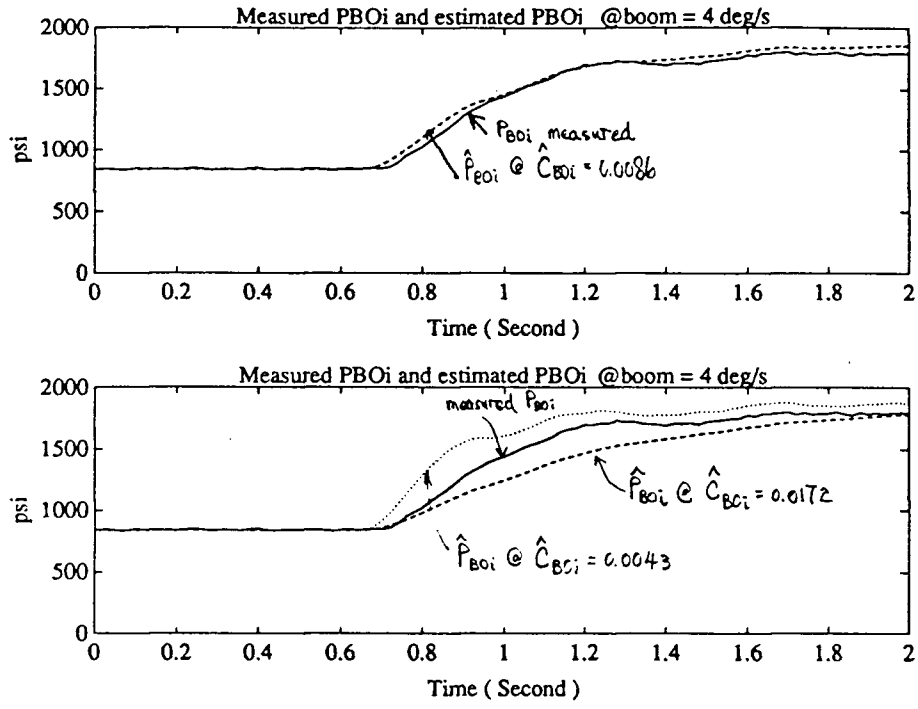


Figure 3.8: Comparison between measured P_{BOi} and calculated P_{BOi}

3.2 Identifying multiple link hydraulic compliance simultaneously

To demonstrate that GA can identify compliance C_{BUi} and C_{BOi} simultaneously, the hydraulic system configuration a, branch #1 is used for illustration (Figure B.1 Appendix B). To test the performance of GA, a simulation program based on the simplified hydraulics model is used (Appendix B). In the simulation, both bucket and boom are activated simultaneously. All the states of the simulated hydraulic system are shown on Figure 3.9a and Figure 3.9b.

Every individual in the population was made up of a binary coded string (20 bits long, 10 bits for each parameter) corresponding to C_{BUi} and C_{BOi}. An individual j was selected from the population pool and its genes $C_{BUi}(j)$ and $C_{BOi}(j)$ were used as follows:

1. At sample data point k, the $\dot{P}_{BUi}(k)$, $\dot{P}_{BOi}(k)$ were obtained from measured $P_{BUi}(k)$, $P_{BOi}(k)$ by linear piece-wise least-square curve fitting technique.
2. $\dot{X}_{BU}(k)$, $\dot{X}_{BO}(k)$ were proportional to the bucket and boom joint angular velocity.

3. $C_{BUi}(j)$ and $C_{BOi}(j)$ were used in Equation B2 (Appendix B) to solve for $\hat{Q}_{BUi}(k)$, $\hat{Q}_{BOi}(k)$. Given the flow $\hat{Q}_{BUi}(k)$, $\hat{Q}_{BOi}(k)$, the internal states $\hat{P}_{11}$, $\hat{P}_{12}$, $\hat{Q}_{11}(k), \hat{Q}_{12}(k)$ were obtained from Equation B3 (Appendix B) .
4. Given that the flow in junction #1 and junction #2 were:

$$\begin{aligned} Q_t &= Q_{BUi} + Q_{11} \qquad Junction\#1 \\ Q_{11} &= Q_{12} + Q_{BOi} \qquad Junction\#2 \end{aligned} \tag{3.10}$$

With the estimated $\hat{Q}_{BUi}(k)$, $\hat{Q}_{BOi}(k)$, $\hat{Q}_{11}(k)$, $\hat{Q}_{12}(k)$ from $C_{BUi}(j)$ and $C_{BOi}(j)$, the summation of flow errors in junction #1 and #2 were used as the fitness criterion for GA.

$$\begin{aligned} Error &= (Junction\#1\ flow\ error) + (Junction\#2\ flow\ error) \\ &= \left|\left(Q_t - \hat{Q}_{BUi}(k) - \hat{Q}_{11}(k)\right)\right| + \left|\left(\hat{Q}_{11}(k) - \hat{Q}_{12}(k) - \hat{Q}_{BOi}(k)\right)\right| \end{aligned} \tag{3.11}$$

5. A window of size 10 was used to minimize the estimated parameter variance and noise effect.

$$Fitness(k) = \sum_{i=0}^{window} bias - Error^2(k-i) \tag{3.12}$$

6. The selection process here was similar to the single link version. If the selected individual j genes $C_{BUi}(j)$ and $C_{BOi}(j)$ were far from the correct value, the Error attached to the string were large and the string was removed from the population. On the other hand, individuals with high fitness values will reproduce more offsprings.
7. For every sample k, GA calculated the internal pressures P_{11}, P_{12}, and the flows Q_{11}, Q_{12}, Q_{BUi}, Q_{BOi} for all the individuals j, $_{(j=1,\ldots,n)}$ in the population and recorded the best individual and its associated internal states.
8. The GA operators (Crossover and mutation) operated on all the binary encoded strings and generate a new population.
9. Step 1 - 8 are repeated until the transient period is over (the line pressures $P_{BUi}(k)$ and $P_{BOi}(k)$ are steady).

In this experiment, the population size was set to 120. The fitness window size was set to 10.

To observe how well GA performed, one can compare GA estimated internal states and the simulated internal states of the hydraulic system (Figure 3.10a and Figure 3.10b). The agreement between both sets of curves is excellent.

Figure 3.11 shows that estimated C_{BUi}, C_{BOi} converge very rapidly to their true value. However, when the generation reaches 425, both estimated C_{BUi}, C_{BOi} start to deviate. This phenomenon is caused by the lack of excitation in the input signal: both simulated $P_{BUi}(k)$ and $P_{BOi}(k)$ approach steady state values and $\dot{P}_{BUi}(k) = 0, \dot{P}_{BOi}(k) = 0$.

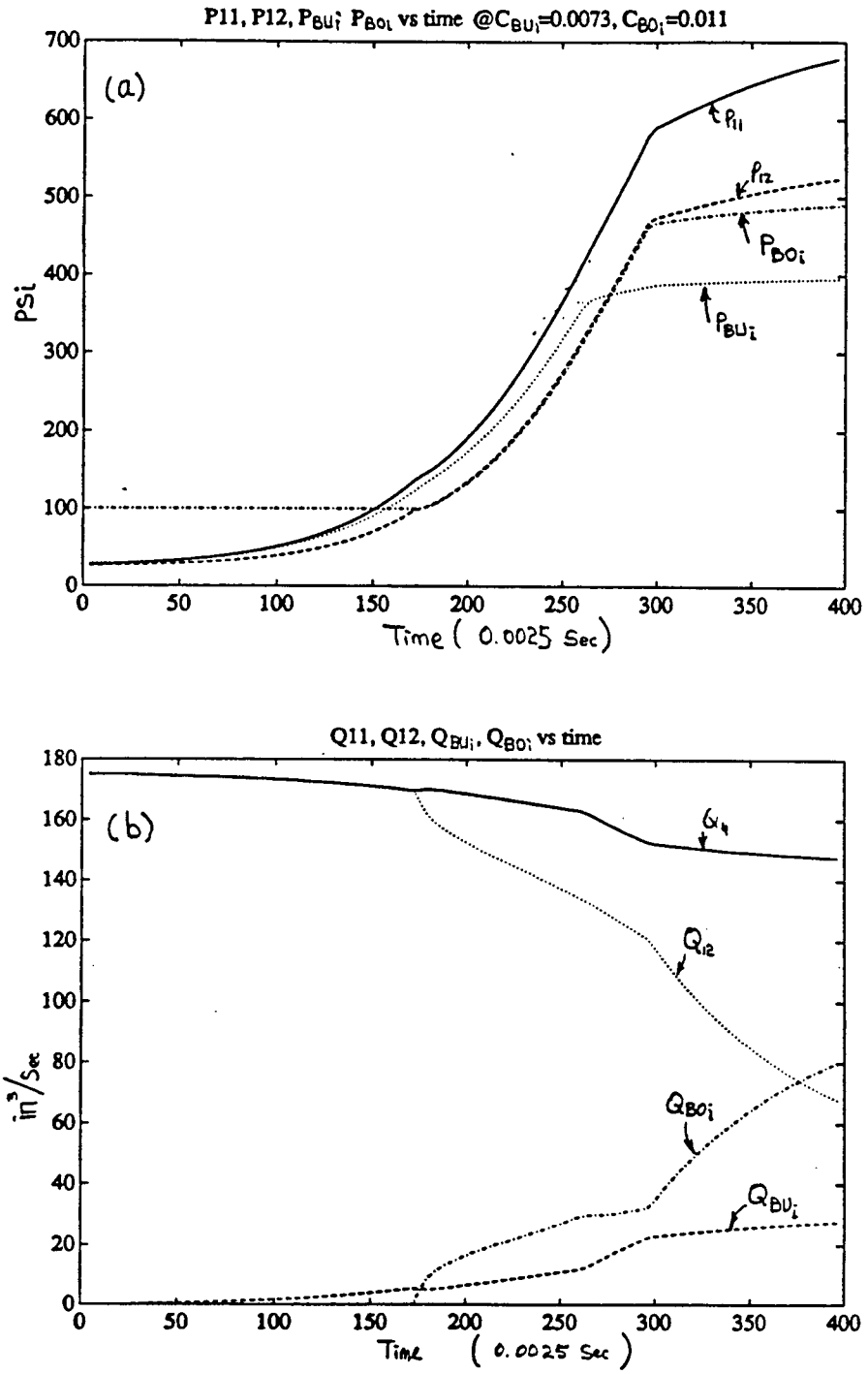


Figure 3.9 Simulation of bucket and boom hydraulic circuit (1 generation = 0.1 Sec)

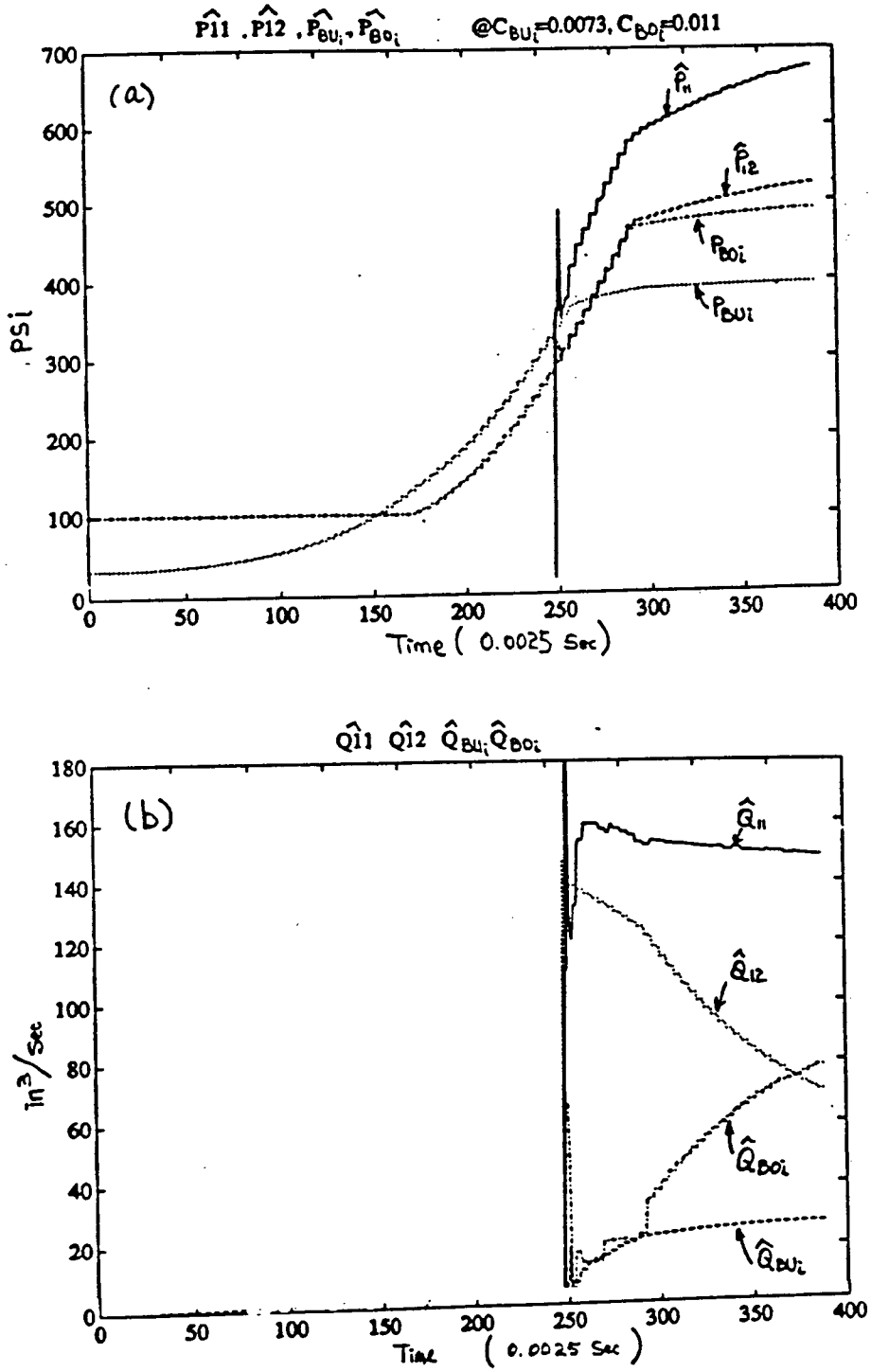


Figure 3.10 Identified internal states from GA (1 generation = 0.1 Sec.)

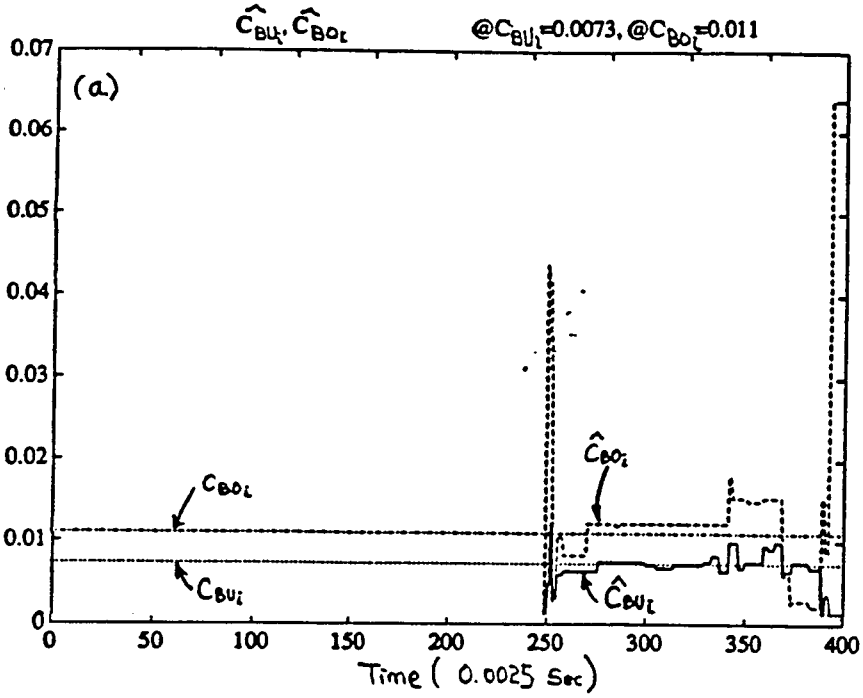


Figure 3.11 Estimated C_{BUi} and C_{BOi} (1 generation = 0.1 Sec)

3.3 On-line identification of hydraulic compliance with GA

Interaction between the Control Algorithm and GA

Depending on the operator commands and the loading of the system, the control algorithm demonstrated in [23] will respond and the hydraulic configuration will change accordingly. Communication between the control algorithm and the diagnostic program is essential since the on-line diagnostic program based on GA needs the configuration of the hydraulic circuit to track and identify the compliance. The on-line diagnostic program should be an integral part of the control system.

The control system determines when and how often to perform a diagnostic check on the hydraulic system. It can also determine which configuration of the hydraulic system and which particular links should be checked by the diagnostic program.

How GA handles multiple configurations of the hydraulic circuit

There are 4 different compliances, C_{BUi}, C_{BOi}, C_{STi}, C_{SWi} that need to be monitored (assume only the pump side of the hydraulic system is monitored). Figure 3.12 shows the interface between GA and the control algorithm. The control algorithm supplies the configuration information to the diagnostic program. The diagnostic program will then run GA with the appropriate set of populations.

Figure 3.13 shows different sets of population for 15 different hydraulic configurations:

If only one link in any configuration is active, then GA will only use population set P1a, or P1b, or P1c, or P1d. For configuration with multiple active links, population sets P2, P3 and P4 will be used. For example, in configuration (a), branch #1 contains the bucket and the boom joints. Assuming that both bucket and boom joints in branch #1 are active, GA must identify the compliance C_{BUi} and C_{BOi} simultaneously. Population set P2a is used because an individual in population set P2a consists of C_{BUi} and C_{BOi} encoded in a binary string, and thus the GA operators such as Crossover, mutation can operate on the encoded parameters $C_{BUi}(j)$ and $C_{BOi}(j)$ simultaneously. If only the bucket is active, then GA will use population set P1a. Note, not all possible 15 combinations of C_{BUi},C_{BOi},C_{STi} and C_{SWi} have to be identified.

Referring to Figure 2.11 in Appendix A, starting from hydraulic configuration (a), two genetic algorithms are used for the two different branches in the hydraulic circuit. GA #1 operates on

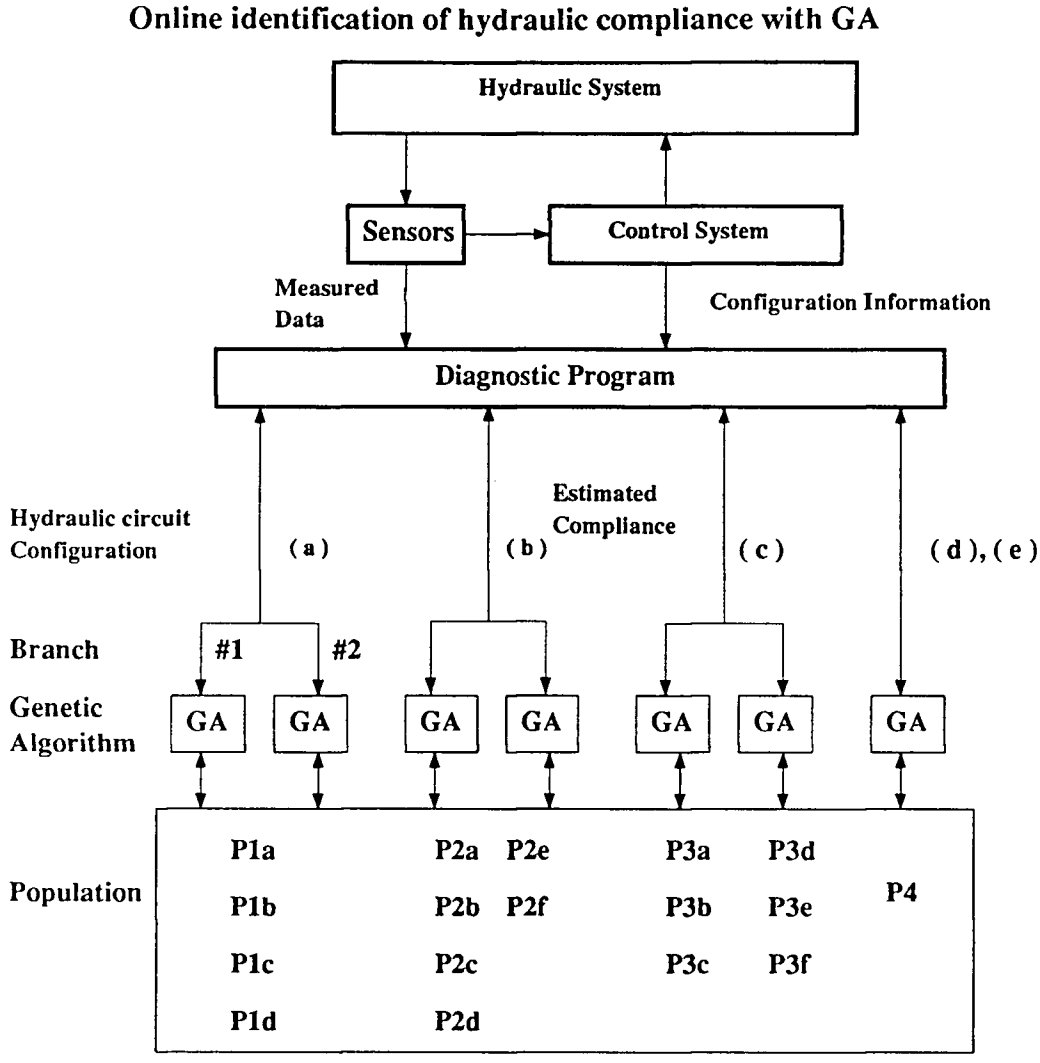


Figure 3.12: Interface between the control system and diagnostic program

population P2a while GA #2 operates on population P2b. If the hydraulic configurations changes from a to b, GA #1 will continue to operate on population P1a, while GA #2 will operate on population P3. Finally, for hydraulic configuration d and e, both branches are connected by the cross-over network. If all the links are active, then all the compliances must be identified at the same time. A GA will operate on population P4 where an individual is a binary string that encodes C_{BUi}, C_{BOi}, C_{STi}, and C_{SWi}.

GA stores the estimated parameters in a population and there are redundant copies of the estimated parameters in different sets of population. For example, C_{BUi} is in population sets P1, P2, P3, and P4. Values of C_{BUi} identified under different population sets correspond to different hydraulic

Population	individual binary encoded string
P1a	C_{BUi}
P1b	C_{BOi}
P1c	C_{STi}
P1d	C_{SWi}
P2a	C_{BUi}, C_{BOi}
P2b	C_{STi}, C_{SWi}
P2c	C_{BUi}, C_{STi}
P2d	C_{BUi}, C_{SWi}
P2e	C_{BOi}, C_{STi}
P2f	C_{BOi}, C_{SWi}
P3a	C_{BUi}, C_{BOi}, C_{STi}
P3b	C_{BUi}, C_{BOi}, C_{SWi}
P3c	C_{BOi}, C_{STi}, C_{SWi}
P3d	C_{BUi}, C_{STi}, C_{SWi}
P4	C_{BUi}, C_{BOi}, C_{STi}, C_{SWi}

Figure 3.13: Different sets of population for different hydraulic configurations

configurations. By combining all the estimated compliances C_{BUi}, C_{BOi}, C_{STi}, C_{SWi} in all different population sets, one can devise:

1. A way to cross check the accuracy of the estimated parameter.
2. A way to identify and to locate the particular faulty condition more precisely.

3.4 Comment and Discussion

Results in the single-link and multi-links experiment indicate:

1. GA can accurately identify the compliance of the boom hydraulic circuit.
2. GA can work with a highly nonlinear system. In the multi-links hydraulic system, not only is the system nonlinear but also there is no closed form solution for the internal states such as P_{11} and P_{12}. GA is able to bypass all the difficulties with its nonrecursive, population based algorithm.

3. GA has unique searching power. It successfully searched out the right combination of C_{BUi} and C_{BOi} from the parameter space in only a few iterations (generations). Such fast parameter convergence rate is very important for real-time identification.

In this chapter, the method of applying GA to identify the compliance of the hydraulic system has been demonstrated. It has also been shown how GA can overcome the nonlinearities and on-line model changes in the hydraulic system. GA has been found to perform exceptionally well in identification of hydraulic parameters.

GA limitations

The GA is similar to other identification methods in that it requires a good model in order to produce accurate result. An inaccurate model of the system under identification may cause failure of the identification process, poor convergence rate, and multiple local minimums.

In discrete parameter identification with the GA, a window size is used to minimize the variance of the parameter and the effect of noise. Because of the secondary effects in the hydraulic system, trying to fit parameters in a large window would result in poor accuracy. From trial and error in the experiments, 10 was found to be the best window size. With a small window size, the variance in the estimated parameter could be quite large due to the effect of noise. Fortunately, in the experiment, the measured P_{BOi}, boom angle were relatively free of noise.

Chapter 4

A Parallel Genetic Algorithm implementation for hydraulic compliance identification

4.1 Introduction

In the previous chapter, the flexibility and searching power of Genetic Algorithms (GAs) in identification of hydraulic parameters have been demonstrated. Das [5] and Kristinsson [12] also achieved good performance using GA in process identification and control. However, the GA differs from other algorithms in that it is based on a population, not on a single point, and it is very computation intensive. Fortunately, the GA is an inherently parallel algorithm. With the advent of parallel processing technology, it is becoming feasible to apply GA in real-time applications.

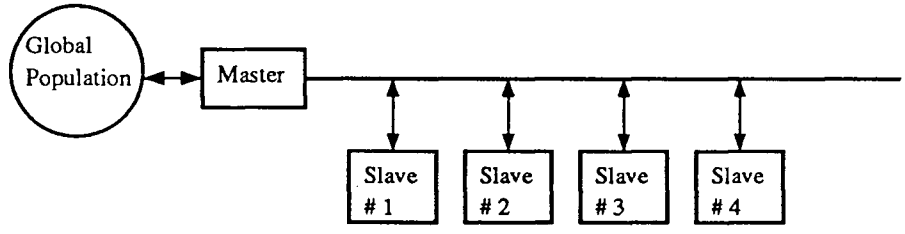
Pettey [16] has suggested some possible parallel GA algorithms. The two most promising ones are: the synchronous master-slave GA (algorithm A) and the cooperating sequential GA (algorithm B) shown in Figure 4.1. Algorithm A is designed for problems which require long evaluation or long fitness calculation time.

Algorithm A is organized in a master-slave configuration with n slaves. The master is responsible for distributing individuals (from a global population) to slaves. After receiving the fitness results from slaves, the master will perform Reproduction and Crossover on the old population to generate a new one. Algorithm A produces exactly the same sequence of generations as the traditional GA, and thus the same solution too. Algorithm A has some disadvantages:

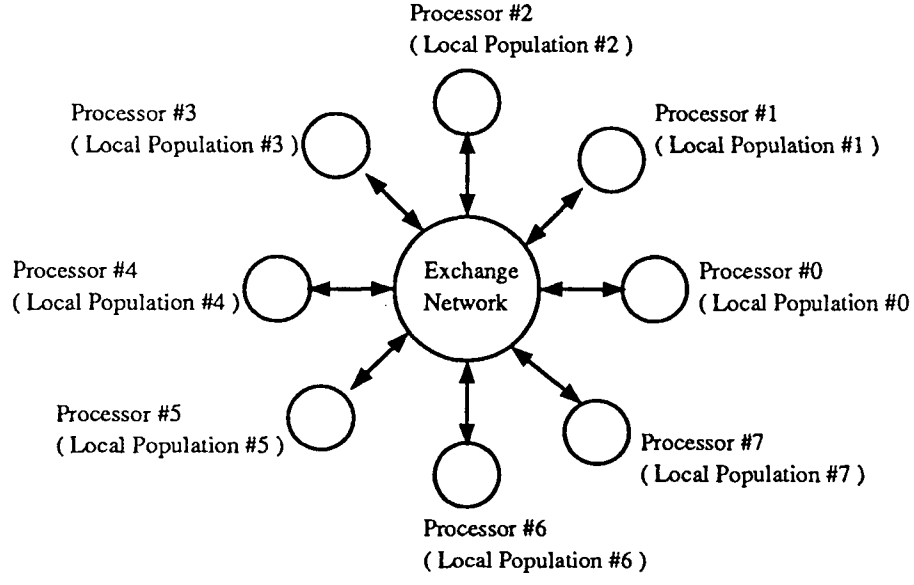
1. Efficiency (defined in Appendix C) per processor decreases as the number of slaves increases since more slaves will be idling while the master performs Reproduction and Crossover.
2. Speedup (defined in Appendix C) is linear for only a small number of slaves.

The cooperating sequential GA (algorithm B) is more generally known since it is the most widely used one. Algorithm B is a collection of many independent (local or nodal) GAs running and communicating with one another simultaneously. Algorithm B gains speed by dividing the global population into many smaller ones for the local GAs. With a smaller population, all local GAs and thus this PGA will run much faster. Algorithm B retains the characteristic of GA by making

all local GAs exchange portions of their population with each other. If the communication time in the exchanging phase is much smaller than the processing time, linear speedup can be achieved. Algorithm B introduces some new parameters: frequency of exchange, portion of the population to be exchanged among NGA, and the local population size. The performance and some characteristics of algorithm B are quite different from the traditional GA. Researchers [17][24] have found that algorithm B is more robust and can consistently find a better solution than the traditional GA in function maximization.



The synchronous master-slave GA (algorithm A)



The cooperating sequential GA (algorithm B)

Figure 4.1: Two Parallel Genetic Algorithms Architectures

In system identification, the fitness function requires a window to minimize the variance of the

estimated parameters and the effect of noise:

$$fitness = \sum_{w=0}^{window\ size} bias - (y(k-w) - \hat{y}(k-w))^2 \qquad (4.1)$$

Where

y(k) : measured system output

$\hat{y}$(k) : estimated system output

bias : a positive constant that sets a limit for GA reproduction operator (only individuals with fitness > 0 can reproduce).

The window in system identification increases the fitness computation cost. From experience, the fitness calculation uses up to 80% - 95% of a GA program execution time, thus satisfying the requirement of algorithm A (Algorithm A is designed for problems which require long evaluation or long fitness calculation time).

Because of the small global population size (100 to 400) [12][5] and the need to use sizable local populations in system identification (explained in the next chapter), the speedup factor from algorithm B is small.

By combining both algorithms A and B in system identification, not only is the overall speedup factor increased, but the robustness of such PGA is improved as well.

Objectives:

A project was initiated with the overall goal to investigate the effect of combining algorithm A and B into a two level parallel implementation. We would like to determine a suitable parallel architecture and its efficiency for such PGA. Such PGA for identification of the hydraulic compliance (single joint) will be implemented on the Inmos T800 Transputer and its performance is compared with that of the traditional GA.

4.2 Speedup, efficiency, and parallel architectures for PGA

The following information from a traditional GA is important for determining the maximum combined speedup factor from algorithm A and algorithm B:

1. The global population size M
2. The parallelizable process (total fitness calculation time for all individuals in the population) t_p

$$\begin{aligned} t_p &= M \times fitness\ calculation\ time \\ &= M \times window\ size \times Error\ calculation\ time \end{aligned} \tag{4.2}$$

3. The sequential process (total GA operator time for all individuals in the population) t_s

$$\begin{aligned} t_s &= M \times GA\ operator\ time \\ &= M \times (Reproduction + Recombination + Mutation)\ time \end{aligned} \tag{4.3}$$

where for system identification, fitness is usually defined as:

$$fitness(k) = \sum_{w=0}^{window\ size} (Bias - Error^2(k-w)) \tag{4.4}$$

Define α as the ratio of execution time of parallelizable process vs the sequential process:

$$\alpha = \frac{t_p}{t_s} = \frac{window\ size \times Error\ calculation\ time}{GA\ operator\ \ time} \tag{4.5}$$

then α is independent of the global population size.

The above equations indicate that there are 2 intrinsic level of parallelisation in GAs. Dividing the global population M into many smaller local populations (M_{loc}) is level 1 parallelisation (algorithm B). Distributing the fitness calculation among slave processors is level 2 parallelisation (algorithm A).

In level 1 parallelisation (algorithm B), the global population is divided into many smaller local populations for local or nodal GAs. The collection of nodal GAs retains the characteristic of traditional Genetic Algorithm by making all local GAs exchange portions of their population with each other. Usually the communication time in level 1 t_{c1} in the exchanging phase is much smaller than level 1 sequential process time t_{s1} and can be neglected. Thus linear speedup over traditional GA is possible. The achievable speedup factor in level 1 is:

$$Speedup_1 = \frac{M}{M_{loc}} \tag{4.6}$$

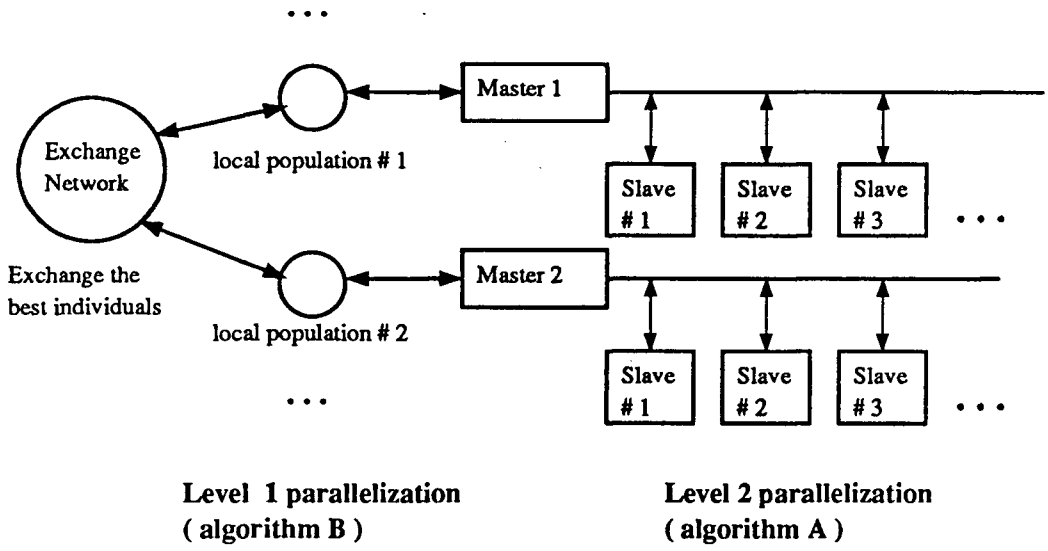


Figure 4.2: Two levels of parallelisation

Where M_{loc} is the local population size and depends on the types of applications.

In level 2, the fitness calculation is distributed among slave processors. Defining:

1. t_{p1} = the parallellizable process timing in Level 1.
2. t_{s1} = the sequential process timing in Level 1.
3. t_{p2} = the parallellizable process timing in Level 2.
4. t_{s2} = the sequential process timing in Level 2.

then the following relationship will be true:

$$\frac{t_{p1}}{t_{s1}} = \frac{t_{p2}}{t_{s2}} = \frac{M_{loc} \times window\ size \times Error\ calculation\ time}{M_{loc} \times GA\ operator\ time} = \alpha \tag{4.7}$$

Even though the ratio of t_{p2} to t_{s2} is equal to α, gain in speedup is possible only if the communication overhead t_{c2} is small compared to t_{s2}. t_{s2} is smaller than t_{s1} since

$$t_{s2} = \frac{M_{loc}}{M} t_{s1} << t_{s1} \tag{4.8}$$

the communication overhead t_{c2} is much more critical in level 2 parallelisation. Interconnection network between master and slave processors must support high speed communication. The speedup

factor of level 2 with n numbers of processors will be (see derivation in Appendix C):

$$Speedup_2 = \frac{t_{p2} + t_{s2}}{\frac{1}{n}t_{p2} + t_{s2} + n \times t_{c2}} = \frac{\alpha + 1}{\frac{\alpha}{n} + 1} \qquad if\ t_{c2} << t_{s2} \tag{4.9}$$

$$Speedup_2 \geq \frac{\alpha + 1}{2} \qquad if\ n = integer(\alpha) \tag{4.10}$$

Where n = number of slave processors

The maximum combined speedup factor for both algorithm A and B (Figure 4.2) will be:

$$Combined\ speedup = speedup_1 \times speedup_2 = \frac{M}{M_{loc}} \times \frac{\alpha + 1}{2} \tag{4.11}$$

The choice of a parallel architecture for PGA

The choice of a parallel architecture for PGA in system identification depends on M, M_{loc} and α. Figure 4.3 shows two possible parallel architectures for PGA.

The hypercube configuration is excellent for applications where

$$\frac{M}{M_{loc}} >> \frac{\alpha + 1}{2} \tag{4.12}$$

Thus only level 1 parallelisation is sufficient. Since the communication overhead is small, speedup is linear for a large number of processors.

For applications where

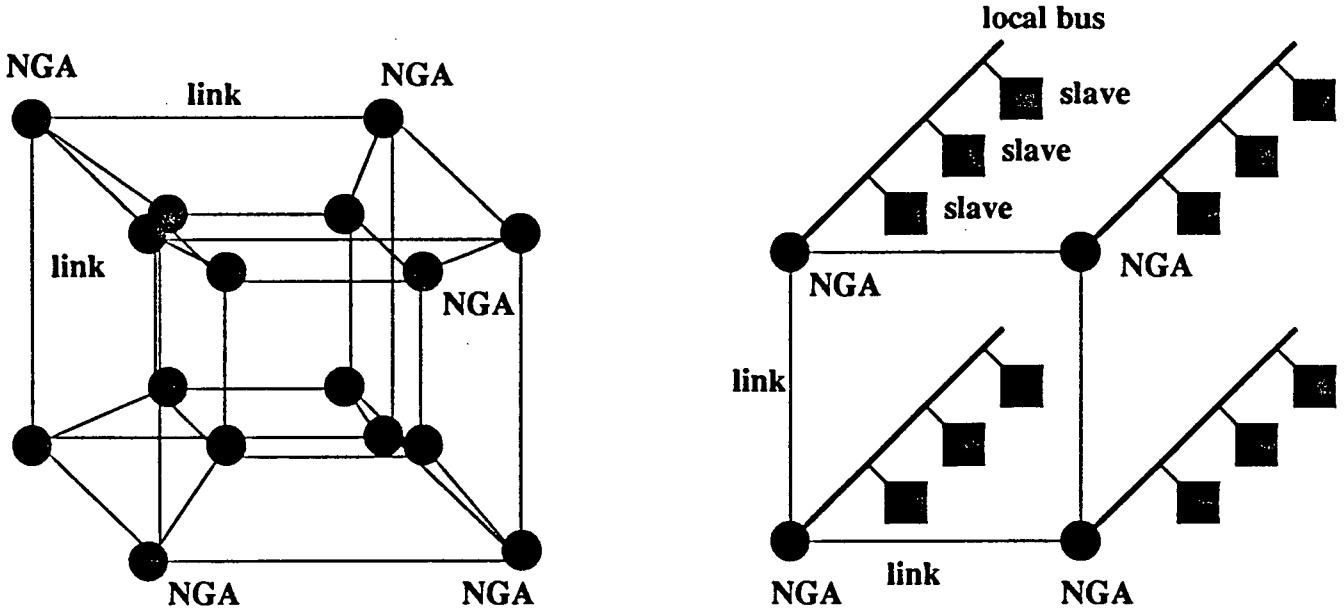
$$\frac{M}{M_{loc}} \approx \frac{\alpha + 1}{2} \tag{4.13}$$

It may be necessary to combine algorithm A and algorithm B to achieve higher speedup factor. A hybrid architecture composed of serial interconnection network for a small number of NGAs and high speed local buses for master-slaves connection can be used. The serial interconnection network handles the nodal GAs (NGAs) communication. The fitness calculations of NGA's local population

are distributed to the slave processes on the local bus. The high speed local bus keeps t_{c2} small to maximize gain in speedup factor.

Efficiency is important in parallel processing. Figure 4.4 shows the software block diagram of a PGA with 2 levels of parallelisation. In this conventional implementation of PGA, the fitness calculation time t_p and GA operators time t_s do not overlap. As more and more processors are added to reduce t_p, the efficiency per processor decreases because more processors are idling during GA operators time. At speedup equal to $\frac{M}{2M_{loc}}(\alpha + 1)$, where t_p is equal to t_s, the efficiency per processor is down to only 50%.

Using pipeline techniques will not increase the maximum achievable speedup factor. However, for a given speedup factor, the number of processors needed is reduced because of improved efficiency (100 %). Figure 4.5 shows the pipeline implementation and the associate timing diagram. By grouping two local GAs together and combining both GAs slave processors for fitness calculation, t_p will be equal to $t_{p1} + t_{p2}$. Buffer 1 holds the temporary result from the fitness calculation corresponds to one of the local populations while a processor performs GA operations on the local population in buffer 2. Such arrangement allows t_p and t_s to overlap each other and forces all slave processors to work at 100 % efficiency. Figure 4.6 and Figure 4.7 show the speedup factors of nonpipeline and pipeline implementation of algorithm A and algorithm B versus number of processors for various α $(8, 16, 24)$ ratio. For small number of processors, the different between the two speedup factors is quite small. However, pipeline implementation offers significant saving in number of processors for higher speedup factor.



Hypercube architecture:
-1 level of parallelization (NGA)
-Small local population (6)
-Linear speedup
-Easy to implement

Hybrid architecture:
-2 levels of parallelization:
population & fitness calculation
-Sizable local population (24)
-Almost linear speedup

Figure 4.3: Two parallel architectures for PGA

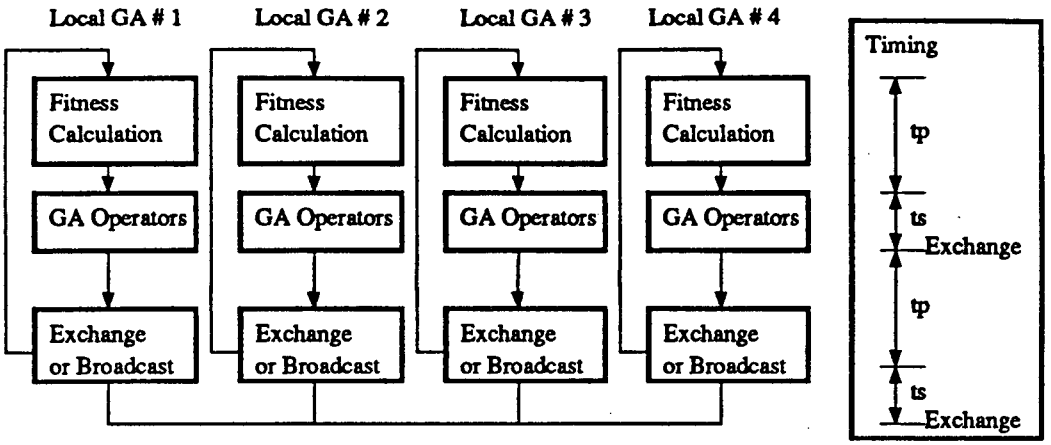


Figure 4.4: Nonpipeline implementation flow chart

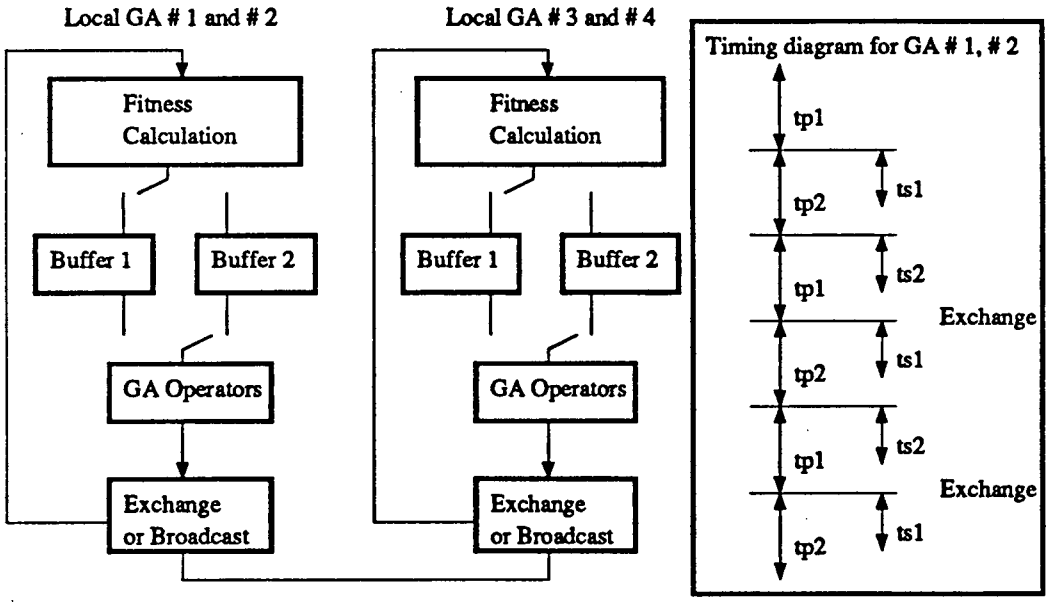


Figure 4.5: Pipeline implementation flow chart

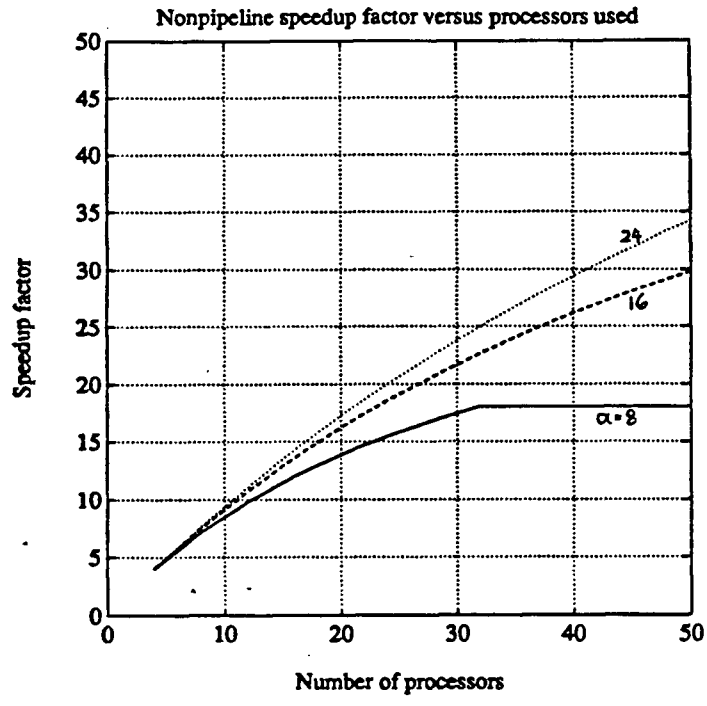


Figure 4.6 Speedup factor of nonpipline implementation versus number of processors

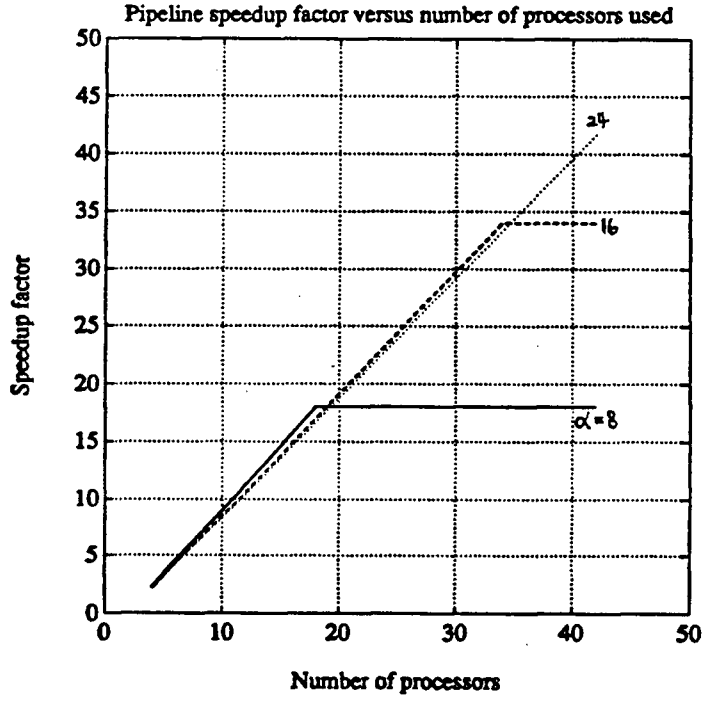


Figure 4.7 Speedup factor of pipeline implementation versus number of processors

4.3 Comparison between PGA and tradition GA performance in identification of hydraulic compliance

A PGA with combined algorithms A and B is used to identify the hydraulic compliance of the boom hydraulic circuit. It is implemented with transputer T800. The block diagram of this PGA and the traditional GA is showed in Figure 4.8. To estimate the hydraulic compliance of boom link, the following parameters are used in the PGA:

1. Global population size = sum of all local population = 120.
2. Local population size = 30.
3. Window size for fitness function = 10.
4. Trial (number of iterations per data sample) = 5.
5. Exchange (best individual) number = 1.
6. Exchange frequency = 1 per 3 generations.
7. GA operators (Mutation, Crossover rate) are identical to traditional GA.

Table 4.1 shows the combined algorithm A and B performance. Note that an impressive speedup factor of 12 is achieved without using any pipelining techniques (for more details on PGA implementation, please consult the appendix).

The data used in PGA experiment is identical to the hydraulic data used in single link compliance estimation. In the PGA experiment, given four local GAs, the estimated compliance of the boom hydraulic circuit is taken from one of local GAs. Figure 4.9a-d and Figure 4.10a-d show the identification process of boom hydraulic compliance with actual spool measurement and indirectly with servo-valve voltage using PGA respectively. In this application, the parameter convergence rate and the estimated C_{BOi} from PGA are almost identical to that from traditional GA.

4.4 Comments and discussion

In system identification, it is possible to combine the synchronous master-slave GA (algorithm A) and the cooperating sequential GA (algorithm B) to increase the speedup factor and robustness of such PGA. Such PGA can be implemented on a hybrid parallel architecture with pipeline techniques.

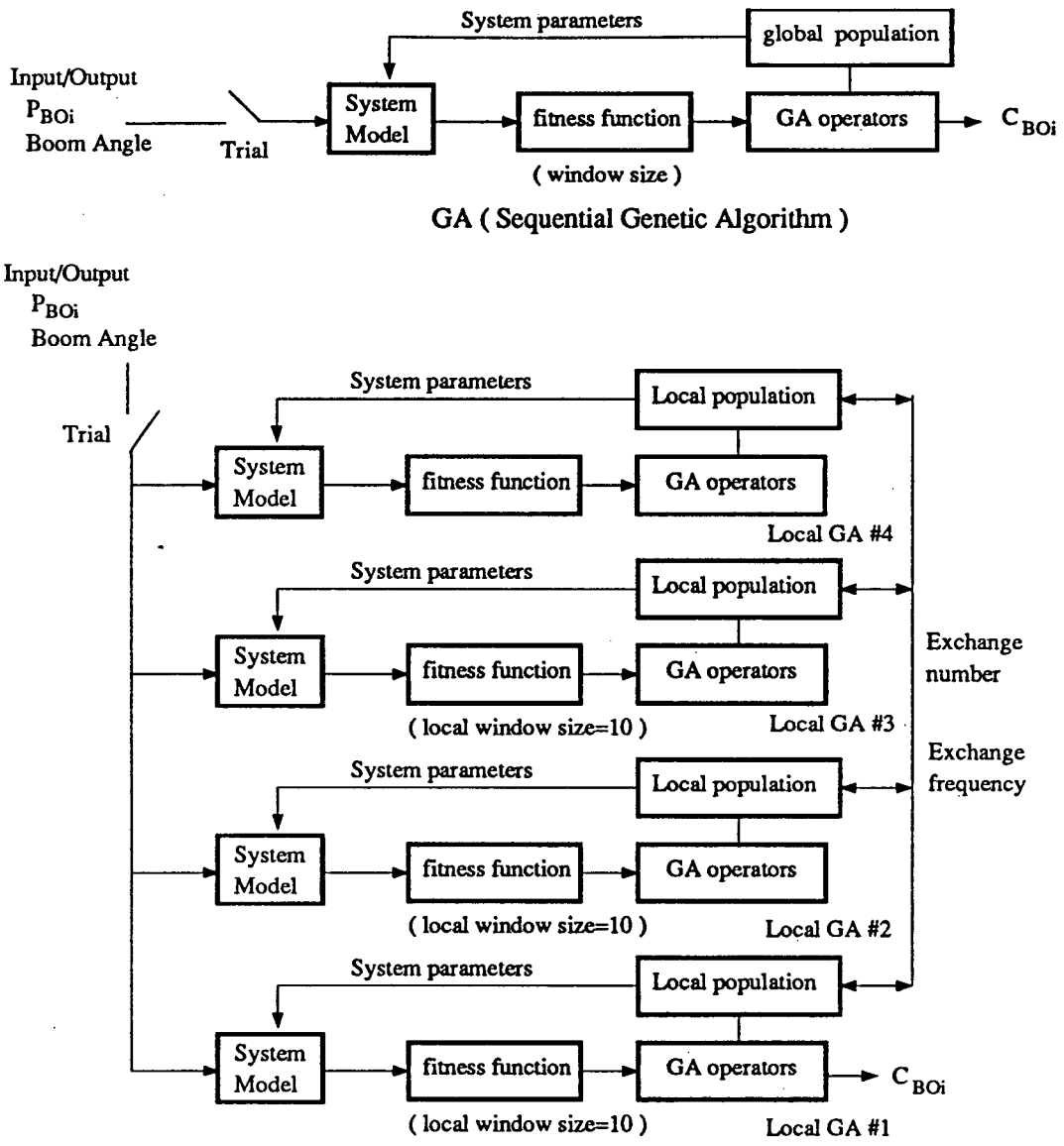


Figure 4.8: Block diagram of GA and PGA for identifying hydraulic compliance

the identification process of boom hydraulic compliance with actual spool measurement and indirectly with servo-valve voltage using PGA respectively. In this application, the parameter convergence rate and the estimated C_{BOi} from PGA are almost identical to that from the traditional GA.

GA and PGA in identifying single link hydraulic compliance :	GA	Level 1 Algorithm (B)	Level 2 Algorithm (A)	Combined Algorithm (A + B)
Number of processors used	1	4	4	16
Global Population size	120	120	N/A	120
Local population size	N/A	30	30	30
Number of generations calculated per sec.	9.8	N/A	N/A	117.6
Speedup achieved	1	4	3	12
Efficiency (Speedup / processors)	1	1	0.74	0.74

Table 4.2 The measured speedup factor of various PGAs

4.4 Comments and discussion

In system identification, it is possible to combine the synchronous master-slave GA (algorithm A) and the cooperating sequential GA (algorithm B) to increase the speedup factor and robustness of such PGA. Such PGA can be implemented on a hybrid parallel architecture with pipeline techniques. The efficiency and speedup factor of such implementation can approach ideal (100% efficiency and linear speedup).

In this application, the communication overhead of PGA is small. Even without direct connections among master and slaves, the communication time is still much smaller than the sequential GA operators time. With an adequate communication network, maximum combined speedup factor of such PGA will be:

$$Maximum\ combined\ speedup = \frac{M}{M_{loc}} \times \frac{\alpha + 1}{2} \tag{4.14}$$

Both GA and PGA have accurately identified compliance of the boom hydraulic circuit. A typical transient interval in which compliance can be identified lasts 0.4 sec ond and contains 20 sample points. From Table E.2, the execution time for traditional GA for 20 sample points can be calculated:

$$\begin{aligned} GA\ time &= sample\ number \times trial \div (generations\ per\ second) \\ &= 20 \times 5 \div 9.8 \approx 10\ second \end{aligned} \tag{4.15}$$

The execution time for PGA will be :

$$\begin{aligned} PGA\ time &= GA\ time \div\ speedup\ factor \\ &= 10 \div 12 \approx 0.8\ second \end{aligned} \tag{4.16}$$

Since PGA-time is twice as long as the transient interval (0.8 second versus 0.4 second), it is necessary to increase the speedup by at least a factor of 2 more.

In the future, T800 Transputer can be replaced by the H9 processor from Inmos Inc. . The H9 processor is about 10 times faster than the T800 Transputer. Real-time identification or control of the hydraulic parameters with PGA will be possible.

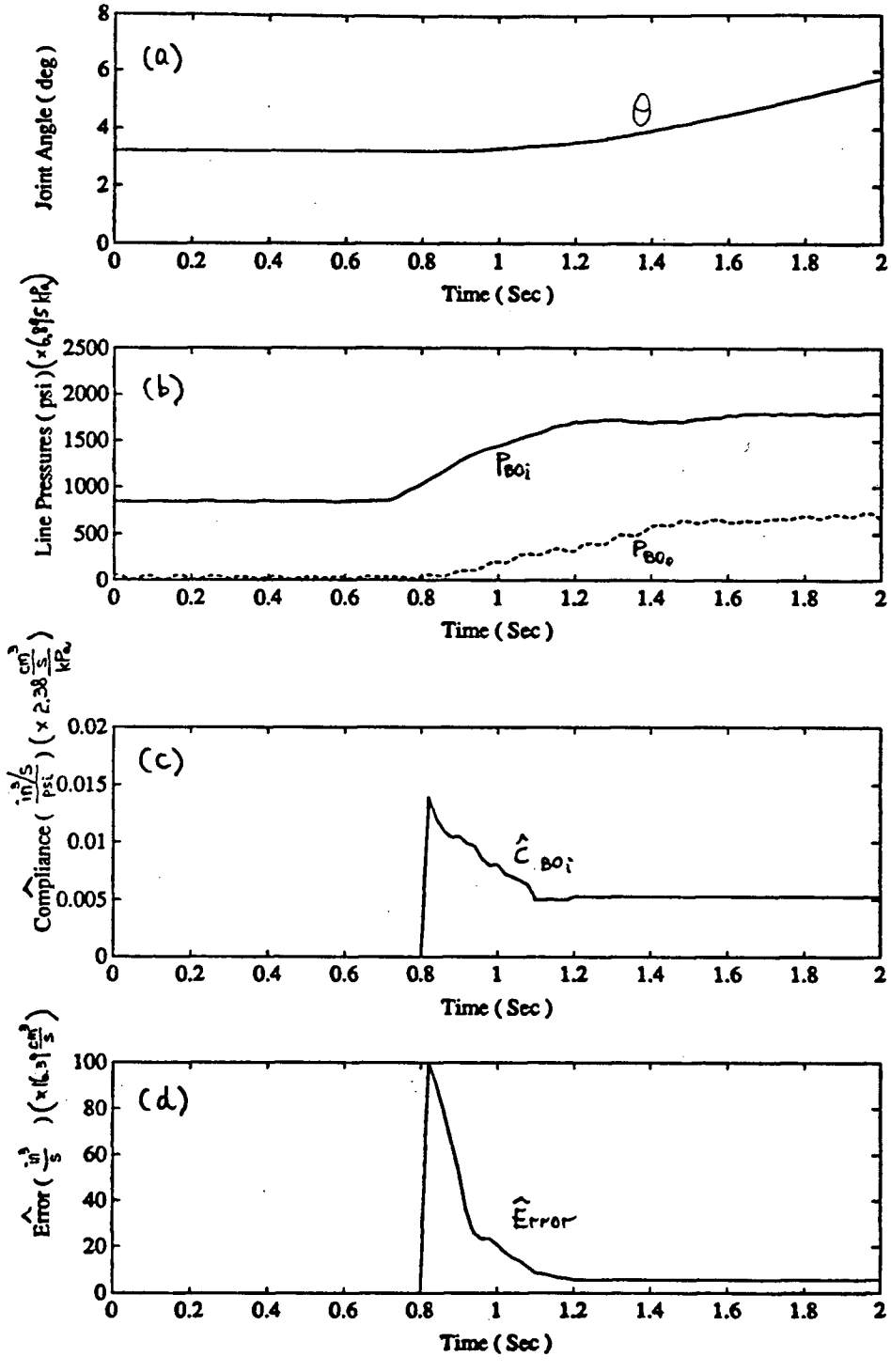


Figure 4.9 Experiment #1 (with spool data)

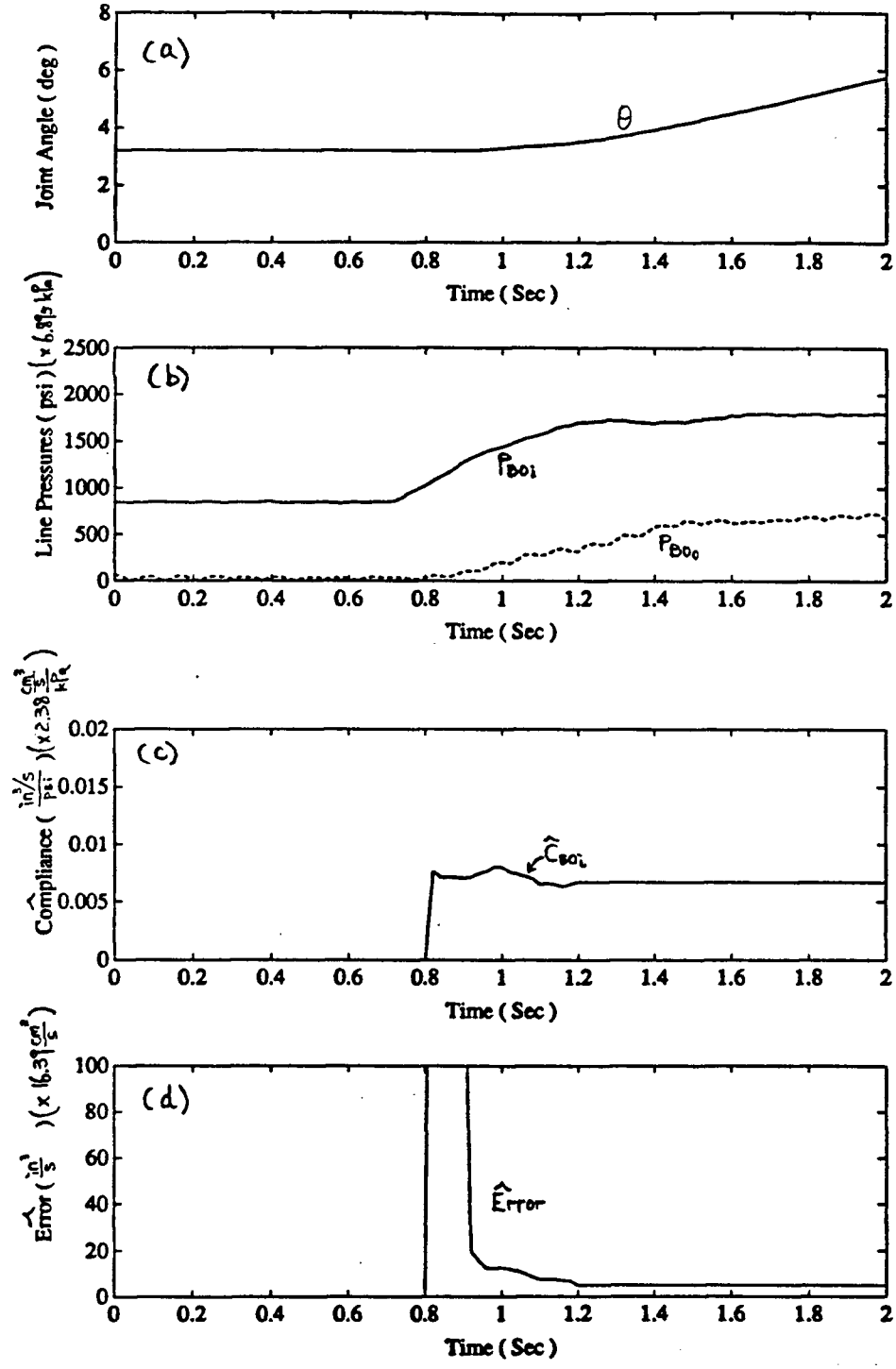


Figure 4.10 Experiment #2 (with servo-voltage data)

Chapter 5

PGA in system identification

5.1 Previous work on PGA (algorithm B)

Pettey, Leuze and Grefenstette[17][16][21] experimented with PGA in function maximization. They implemented a PGA on a hypercube processor network with a local population size of 50. Their experimental results agreed well with Goldberg's paper [9] on the optimum global population size. In [21], based on the assumption of uniformly distributed communication among NGAs, Pettey suggested that PGA maintains efficiency similar to that observed in a traditional sequential GA.

Tanese [24] implemented a PGA on a medium-grained hypercube computer for function maximization. He experimented with a number of important parameters such as exchange frequency, number of individuals exchanged, Mutation and Crossover rates, and local population size. He suggested that:

1. Exchange frequency and the number of individuals exchanged depend on the problem domain. Exchanging too frequently or too infrequently may degrade PGA performance.
2. Different Mutation and Crossover rates may be used among nodal GAs (NGAs). A NGA with a high Mutation rate would emphasize exploration and a NGA with low Mutation rate exploitation.
3. The local population size can be as small as 6 without affecting the measured parameter convergence rate.

Previous works on PGA (algorithm B) can be summarized as follow:

1. Global and local population size of PGA depends on applications.
2. PGA is consistently able to find a better solution than the sequential GA[25].
3. PGA may be more robust than sequential GA. [24]
4. PGA parameter convergence rate may be slower than the sequential GA running on the same population. A super-individual is seen only by a local population and it takes several exchanges before such a super-individual can be seen by the global population.

Objectives:

While many experiments have been done on function maximization with GA and PGA, there are very few works [5][13]on using PGA in system identification and control. It is in this area that we would explore the full potential of PGA in system identification.

5.2 Important requirements on PGA in system identification

5.2.1 The conventional PGA implementation (algorithm B)

In a conventional PGA, each processor runs a genetic algorithm on its own local population, periodically selects good individuals from its local population, and sends copies of good individuals to one of its nearest neighbours. Under such an exchange scheme, the local GAs are best connected in a hypercube multiprocessor network. The hypercube network is showed in Figure 4.3. There are many implementations of PGA on the hypercube network [24][17]. By sequentially sending and receiving just one best individual from one of its nearest neighbours, each processor on the hypercube network can keep the ratio of local population size to incoming external member sufficiently high to maintain the local population diversity. In [24], a PGA was implemented on a hypercube network and the smallest local population size was 6. If the hypercube network is used, by dividing the global population (100) into many small local populations (6), PGA can achieve a speedup factor of 16 time over sequential GA.

For a PGA with small local population (6), it is necessary not to broadcast any "best individual" from one local GA to the others. Such an individual may destroy the diversity of all the local populations (a best individual may be a local minimum). Sequential exchange of good individuals among local GAs is necessary.

In the case of a super individual (global minimum), depending on the hypercube network, it will take several exchanges before the entire population will be aware of such super individual. Thus such an exchanging scheme will cause the total population to converge at a slower rate than sequential GA, since a "super individual" would be seen by subpopulations instead of the global population.

From experience in working with simple PGA exchange mechanisms, with the diversity of the population preserved, a slower convergence rate avoids premature convergence and leads to a global minimum. For conventional PGA, such a trade off between the robustness and the parameter convergence rate is unavoidable.

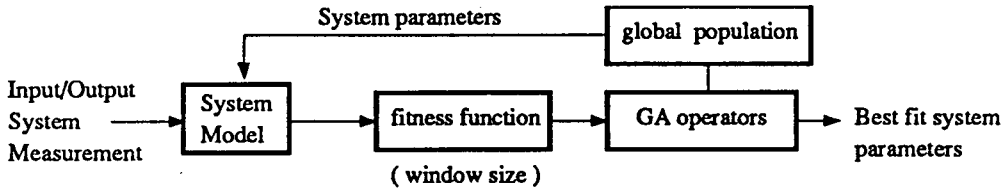
Many papers have been published in the last few years on PGA performance in function maximization [17][24]. In function maximization, PGAs have been shown to be more robust than the sequential GA. Even though there have been many successful implementations of PGA for function maximization, the requirements and the nature of the problem in system identification are quite different from those in function maximization.

5.2.2 Differences between function maximization and system identification

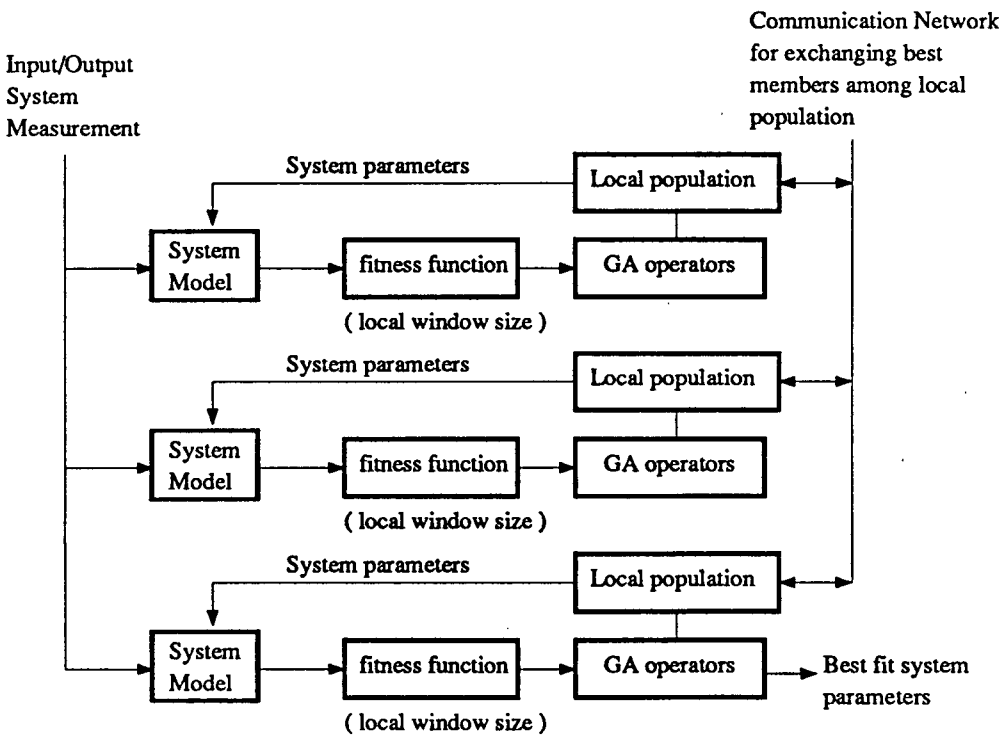
There are many differences between function maximization and system identification for PGA:

1. In function maximization, the environment is static; the function is time invariant. In system identification, the working environment is usually dynamic and in real-time.
2. In function maximization, it is possible to use different Mutation rates among nodal GAs (NGAs) so that some NGAs will place emphasis on exploration and others on exploitation [24]. However a higher Mutation Rate will increase the variance of the estimated parameter. In function maximization, where only the final value is needed, higher variance in parameters is not important. But in many applications of system identification and control, where tracking the parameter changes is important for control and diagnosis, the variance of parameters must be minimized. Therefore only a low Mutation Rate should be used in system identification.
3. In function maximization, the global population size plays a significant role in determining the parameter convergence rate (measured with respect to time) [17]. In system identification, the parameter convergence also depends on data sample rate and excitation in the sampled signal. Thus the role the global population plays is secondary in system identification.
4. In real-time system identification and control, a high parameter convergence rate is one of the most important requirements.

Figure 5.1 shows the block diagram of GA and PGA for system identification. Several important areas in the figure are: the global and local population size, the communication network and the fitness function.



GA (Sequential Genetic Algorithm)



PGA (Parallel Implementation of GA)

Figure 5.1: Block diagram of GA and PGA in system identification

Global population size

In function maximization, according to Goldberg [7][8] , the size of the global population should depend on the length of an individual string. Using Goldberg's criteria, very large global population

sizes, from 500 to 1000 are usually used in function maximization. R. Das [5] and K. Kristinsson [12] demonstrated that in system identification with a small number of parameters (4 – 6) with GA, good performance can be obtained from small global population size of 100. Such a result is encouraging since the computation cost is directly proportional to the global population size. In parallel implementation of Genetic Algorithm (PGA), it will be advantageous to keep the same small global population size of 100. Only if the performance of PGA with such a small global population size is unsatisfactorary, then more local GAs can be added to increase the global population.

Local population size

In a static environment, Tanese[24] demonstrated that a very small local population size of 6 could be used in his application. For system identification, it is necessary to use a larger local population. In a dynamic environment, diversity of small local populations can be destroyed easily. For example, when the parameters of the system under identification become constant, then many small local populations may become identical or very similar (low Mutation rate is used). Larger local population offers better protection.

Based on the simulations and the above considerations, PGA's local population size for system identification was selected to be at least 20.

Communication network

In real-time system identification, the measured input and output of the system must be broadcasted to every local GA. The communication network should have very small and deterministic delays in broadcasting. Most PGAs mentioned in the literature were used in the area of function maximization and did not require much communication among local GAs. Depending on the speed of the processor, applications, and sampling rate, a high speed communication network among NGAs is needed in some real-time applications.

5.3 PGA performance Improvement in system identification

5.3.1 Distributed fitness function

In discrete time parameter identification, GA's fitness function requires a window to minimize the variance of the estimated parameters and the effect of noise:

$$fitness = \sum_{w=0}^{window\ size} bias - (y(k-w) - \hat{y}(k-w))^2 \qquad (5.1)$$

Where

y(k) = measured system output

$\hat{y}(k)$ = estimated system output

bias = a positive constant that set a limit for GA reproduction operator (only individual with fitness > 0 can reproduce).

window size = used to minimize the estimated parameter variance.

GA with a large window size will be more immune to noise, but will be slower in tracking changes in parameters. Conversely, GA with a small window size may perform poorly in noise, but is excellent in tracking changes. With multiple local GAs exchanging their best individuals, PGA may perform better if the local GAs use different window sizes. A local GA with large window size could estimate parameters with good immunity to noise, and a local GA with smaller window size may track changes in the system parameters better. In exchanging their best individuals, both local GAs may complement and compensate each other's advantages and disadvantages. Thus a PGA with such settings may improve its parameter convergence rate and maintain a good noise immunity at the same time .

The robustness of the PGA, and to some extent, the PGA parameter convergence rate, can be improved by increasing the global population size. However, for a given sample rate, increasing the global population will require more computation power. With different window sizes on the local GAs, a local GA with smaller window size has a smaller computational load. Thus it is feasible to increase the population of such local GA to obtain the same computation load. With a smaller window size and a larger local population, the tracking performance of such local GA is further enhanced.

By varying the window size and local population accordingly to maintain the same computation load, the global population of PGA is increased and the robustness of PGA is also improved. The term distributed fitness function is used to describe the combination of different fitness function and local populations.

In real-time system identification and control, the ability to track changes in the system parameters (a fast parameter convergence rate) is essential.

5.3.2 Broadcasting with active error analysis

Instead of exchanging the best individual with only one of the neighbours, all local GAs will periodically broadcast their best individuals to one another. Broadcasting has the following advantages over sequential exchange:

1. Since all local GAs receive the best individuals from one another, it is equivalent to having every local GA sees the entire global population. The parameter convergence rate will be comparable to that of sequential GA.
2. Broadcasting allows the distributed fitness function's full potential to be exploited. Based on the Maximum Likelihood technique, extra information from the distributed fitness function can be analyzed and be used in a more active role in estimating the unknown system parameters.

During the exchange phase, broadcasting ensures that every local GA receives copies of the best individuals from other local GAs with different window sizes. One generation later, every local GA would calculate the fitness value for its own population, which also includes the best individuals from other local GAs. Assuming there are 4 different local GAs with window sizes of 30, 24, 16, 10 respectively, a matrix of fitness of best parameters versus different window size can be extracted from the local GAs with no computation cost:

window size					Sw_j
w_1=30	Error(w_1, A_1)	Error(w_1, A_2)	Error(w_1, A_3)	Error(w_1, A_4)	Sw_1
w_2=24	Error(w_2, A_1)	Error(w_2, A_2)	Error(w_2, A_3)	Error(w_2, A_4)	Sw_2
w_3=16	Error(w_3, A_1)	Error(w_3, A_2)	Error(w_3, A_3)	Error(w_3, A_4)	Sw_3
w_4=10	Error(w_4, A_1)	Error(w_4, A_2)	Error(w_4, A_3)	Error(w_4, A_4)	Sw_4
SA_i	SA_1	SA_2	SA_3	SA_4	

Figure 5.2: A matrix of fitness error versus different window size

Where

$$\begin{aligned} &\textit{Given a local GA with window } w_i : \\ &fitness(w_i, A_j) = \sum_{n=0}^{w_i} \left(B - \left(y(k-n) - \hat{y}(k-n]_{A_j}\right)\right) \\ &Error(w_i, A_j) = \sum_{n=0}^{w_i} \left(y(k-n) - \hat{y}(k-n]_{A_j}\right) \\ &A_j = \textit{parameters from a local GA with window } w_j \end{aligned} \tag{5.2}$$

$$\begin{aligned} &SA_j = \sum_{i=1}^{4} \left(\frac{|\ \mathrm{Error}(w_i, A_j)\ |}{|\ \mathrm{Error}(w_j, A_j)\ |} \right) \\ &= \sum \text{normalized Error for parameter } A_j \text{ (derived from } w_j) \\ &\text{evaluated under all other window sizes : } w_{i\ (i=1..4)} \end{aligned} \tag{5.3}$$

$$\begin{aligned} &Sw_i = \sum_{j=1}^{4} \left(\frac{|\ \mathrm{Error}\ (w_i, A_j)\ |}{|\ \mathrm{Error}\ (w_j, A_j)\ |} \right) \\ &= \sum \text{normalized Error for parameters } A_{j\ (j=1,2,3,4 \text{ window size})} \\ &\text{evaluated under window } w_i \end{aligned}$$

For example:

$$\begin{aligned} Sw_1 &= \left|\frac{Error\ (w_1, A_1)}{Error\ (w_1, A_1)}\right| + \left|\frac{Error\ (w_1, A_2)}{Error\ (w_2, A_2)}\right| + \left|\frac{Error\ (w_1, A_3)}{Error\ (w_3, A_3)}\right| + \left|\frac{Error\ (w_1, A_4)}{Error\ (w_4, A_4)}\right| \\ SA_1 &= \frac{|Error\ (w_1, A_1)| + |Error\ (w_2, A_1)| + |Error\ (w_3, A_1)| + |Error\ (w_4, A_1)|}{|Error\ (w_1, A_1)|} \end{aligned} \tag{5.4}$$

Active error analysis:

From the definition of Sa_j and Sw_i, the following statements can be made:

1. Under a noiseless and transitional environment (i.e. a system parameter is changing) the local GA with smallest window would be first to track such change. During the transient time, Error(w_4, A_4) would be smaller than Error(w_4, A_3), Error(w_4, A_2), Error(w_4, A_1). From the definition for Sw_j, Sw_4 will be larger than Sw_1, Sw_2, and Sw_3 with high probability:

$$\begin{aligned} &Noiseless\ and\ transitional: \\ &Sw_4 > Sw_3 > Sw_2 > Sw_1 \ \ with\ high\ probability \\ &where: \\ &A_4 = estimated\ parameter\ from\ GA\ with\ w_4 \\ &A_3... \\ &A_2... \\ &A_1 = estimated\ parameter\ from\ GA\ with\ w_1 \end{aligned} \tag{5.5}$$

   Thus the magnitude of Sw_j indicates which window is most suitable for the present time.

2. In a noisy but steady-state condition, the summation of Error across all window sizes should indicate which estimated parameter ai has the best overall Error. Since the variance of the estimated parameter in window 30 is smallest, Error(w_1, A_1) will be less than Error(w_2, A_2), Error(w_3, A_3), Error(w_4, A_4). In this case, SA_1 should be less than SA_2, SA_3, and SA_4.

$$\begin{aligned} &Noise\ and\ steady\ state: \\ &SA_1 < SA_2 < SA_3 < SA_4 \ \ with\ high\ probability \\ &where:\ w_1 > w_2 > w_3 > w_4 \ \ (window\ size) \end{aligned} \tag{5.6}$$

3. The information in SA_i and Sw_i can be used in choosing the best parameter from the four different windows under noisy, noiseless and transitional environment. There are many ways to use the information in Sw_i and SA_i. One obvious way is to use the ratio Sw_i/SA_i as a selection criteria. The ratio Sw_i/SA_i will compensate for some of the noise effect and account for the

transient condition. The highest ratio of Sw_i/SA_i is used to select the best parameter from the window size i.

Under most conditions :

$\frac{Sw_i}{SA_i}$ $_{(i=1...4)}$ *is a good indication of which window is most appropriate*

For example :

$$if\ \frac{Sw_2}{SA_2} > \frac{Sw_1}{SA_1}, \frac{Sw_3}{SA_3}, \frac{Sw_4}{SA_4}\ \ then\ window\ w_2\ is\ \textbf{probably}\ the\ most\ appropriate\ one \tag{5.7}$$

The computation cost for the active error analysis is insignificant.

By combining the distributed fitness function with the active error analysis, it may be possible to improve PGA performance in parameter convergence rate, noise immunity such that they are equal or better than the sequential GA.

The active error analysis will output the estimated parameters at the broadcasting frequency.

Disadvantage of broadcasting:

Broadcasting the best individuals among local GAs could destroy diversity of the local population. Several measures have to be taken to avoid the destruction of local diversity that could lead to premature convergence:

1. The local population size should not be smaller than 20 (mentioned in previous sections) to maintain the high ratio between local population size and incoming exchange individuals (6 : 1).
2. The broadcasting frequency should be kept low (less than 1 per 10 generations)

The distributed fitness function will provide additional protection on the local population diversity. Although higher Mutation rates can also be used to protect the diversity of local population too, it does severely affect the variance of the estimated parameters. Also, since Mutation is a random process, it is difficult to determine the effectiveness of such mechanism. Thus a **higher** Mutation rates will not be used unless all other methods have failed.

5.4 Testing the distributed fitness function and active error analysis on various PGA configurations

In GA and PGA, the initial population is usually created by a random number generator. Thus the initial parameter convergence rate depends on the population size and the random number generator seed. The dependency becomes more noticeable in PGA since each local GA has its own population size and random number generator seeds. For example, a good individual in the initial population will distort the measurement of the parameter convergence rate. Some of the distortion can be removed by averaging many runs of GA or PGA to give a better estimate of the parameter convergence rate. In discrete time parameter identification, the interest is not only on the initial parameter convergence rate, but also on how well PGA can track the changes in the system parameters.

A second-order system of the following form (the same one in [12]):

$$y(k) = a_1 * y(k-1) + a_2 * y(k-2) + b_0(u(k) + b_1 * u(k-1)) \tag{5.8}$$

Where a_1=1.5, a_2=-0.7, b_0=1.0 b_1=0.5.

is used to test the performance of various PGA configurations. Because GAs do not require linearity in the parameters, they can directly identify the poles and zeros of the system. In pole-zero form, the plant can be written as:

$$A(q^{-1})y(t) = B(q^{-1})u(t-d) + C(q^{-1})e(t) \tag{5.9}$$

$$Where:$$

$$A(q^{-1}) = (1 - (\alpha_1 \pm j\beta_1)q^{-1})$$

$$B(q^{-1}) = b_o(1 - (\gamma_1 \pm j\delta_1)q^{-1})$$

$$d = delay$$

$$e(t) = noise$$

For a stable minimum phase plant, the poles and zeros are inside the unit circle. Therefore the search space can be limited to the unit circle or a box enclosing the unit circle. With GA, the prior

	lower bound	upper bound	# of bits	resolution
d	1	4	2	1
bo	0.0	2.0	7	0.016
α_1,β_1, γ_1, δ_1	-1.0	1.0	7	0.016

Table 5.1 Search space for a stable minimum phase system

PGA	Nodal GA	Exchange (E) Broadcast (B)	Comment
Configuration #1	4	E : 1 per every 3 generations	Use sequential exchange
Configuration #2	4	B : 1 per every 12 generations	Use broadcast

Table 5.2 Two different PGA configurations

knowledge of the exact size of the search space is an important advantage of using pole-zero form instead of the parameter form.

The second order system has six parameters that need to be identified: delay d, gain b_o and poles-zeros α_1, β_1, γ_1, δ_1 . The delay is coded as a two bit string to give 4 choices and the other parameters are coded as five 7–bit strings, making totally a 37 bit string. This results in a search space with total $2^{37}=1.371*10^{11}$ combinations. The parameters have been concatenated as follows $d \mid \alpha_1 \mid \beta_1 \mid \gamma_1 \mid \delta_1 \mid b_0$. Upper and lower bounds on the parameters are defined and the resolution of the coding is in Table 5.1

A PRBS (Pseudo Random Bit Sequence) signal[12] is used as an input for the system. The PRBS input has a period of 127 with the bit interval equal to four times the sampling interval (PRBS signal is persistently excited signal) .

Unless stated otherwise, the genetic operator settings are :

1. The Crossover rate (p_c) = 0.8
2. The Mutation rate (p_m) = 0.01

Two PGA configurations are tested: the PGA configuration #1 and the PGA configuration #2 in Table 5.2.

The experiment setup is listed in Table 5.3. In experiment #1, the effectiveness of the distributed fitness function with different local population sizes and fitness window sizes on a PGA using sequential exchange mechanism is tested. Experiment #2 is designed to test whether the distributed fitness function can effectively protect the local population diversity against broadcasting. The highest broadcasting frequency is used (1 per generation), and the performance of different local GAs is compared. Experiment #3 and #4 are designed to test how the combination of distributed fitness function and active error analysis can help to improve the performance of PGA in system identification. Finally, the performance of PGA and GA is compared in experiment #5.

Experiment	Fitness Function	Number of local GAs	Local population size	Fitness window size	B=Broadcast E=Exchange
#1 (PGA)	traditional	4	24:24:24:24	30:30:30:30	E = 1 per every 3 generations
	distributed		24:30:40:50	30:24:16:10	
#2 (PGA)	traditional	4	24:24:24:24	30:30:30:30	B = 1 per every generation
	distributed		24:30:40:50	30:24:16:10	
#3 (PGA)	traditional	4	24:24:24:24	30:30:30:30	B = 1 per every 12 generations
	distributed		24:30:40:50	30:24:16:10	
#4 (PGA)	distributed + active error analysis	4	24:30:40:50	30:24:16:10	B = 1 per every 12 generations
#5 (GA)	traditional	1	100	30	N/A

Table 5.3 Experiment setup

In Table 5.3 , notation such as 24:30:40:50 under column heading "Local population size" implys that local GAs #1, #2, #3, #4 population size are 24, 30, 40, 50 respectively. To observe how well PGA tracks changes in the system parameters, the parameter a_1 of the second order system is altered as shown in Table 5.4.

Case	0-199 generation	200-399 generation	400-600 generation
1	a_1=-1.5	a_1=-1.5+1.0*(gen-200)	a_1=-0.5

Table 5.4 How a_1 of the second order system is altered

5.4.1 Experimental result

Initially the fitness value was assumed to be a good indication of how well GA can track the system parameters. However, further examination showed that sometimes a small fitness error only indicates good tracking on the output of the system, and is not necessarily an indication of the convergence of the estimated parameters. A good example would be the fitness error from a GA with a window size of 10. Even though the fitness error was small in the presence of noise, the variance of the estimated parameter can be very large. Thus fitness error alone would not indicate the performance of various PGA configurations.

In all the experiments, various PGA configurations were used to identify the pole and zero of the second order system. The estimated real and imaginary part of the dominant second order system pole (α_1, $j\beta_1$) from various PGA configurations were plotted.

Experiment #1: Testing the performance of the PGA configuration #1 with the distributed fitness function

PGA configuration #1 based on a simple 2 dimensional hypercube (4 processors) was implemented to test whether the distributed fitness function could improve performance. The experimental results are shown in Figure 5.3a-d. The variance of the estimated parameter seemed to be reduced slightly (compare to Figure 5.3a and Figure 5.3c), but there is no other indication of any significant improvement. The passive exchange mechanism of the PGA configuration #1 could not utilize the full potential of the distributed fitness function.

Experiment #2: Testing the effectiveness of the distributed fitness function in protecting the local population diversity

For the PGA configuration #2, the local population diversity was threatened by the broadcasting mechanism. The distributed fitness function with a different local population and window size, should give some protection to the local population diversity. In the experiment, the PGA configuration #2

with the same fitness and distributed fitness function were subjected to a broadcasting frequency of one per generation. Under such a high broadcasting frequency, without sufficient protection, the local population diversity would be destroyed and led to premature convergence. Figure 5.4a and Figure 5.4b show two local GAs from the PGA configuration #2 with the same fitness function performed. After 200 generations, the estimated poles from two different local GAs became identical and converged prematurely. The performance of the PGA configuration #2 with the distributed fitness function, is shown in Figure 5.4c and Figure 5.4d . After 400 generations, the estimated poles from the two local GAs were close but remained different. Even though the convergence rate of the estimated pole was much slower than normal, there was no indication of premature convergence. The distributed fitness function had successfully preserved the local population diversity.

Experiment #3: Testing the performance of the PGA configuration #2 with the distributed fitness

Figure 5.5 shows how the real and imaginary part of the pole changed with generation (time). Window size affected the GA parameter tracking ability. In a noiseless environment, the local GA with a window size of 10 could track the change in the pole better than one with a window size of 30. Given that the pole started to change at 200 generation, the local GAs with window size 10 and 30 started responding to such change at 250 generation and 300 generation respectively. The different delay in the response was caused by the local GA different window size. (Figure 5.5c and Figure 5.5d). In a noisy environment, the local GA with a window size 30 could filter out noise much better than the GA with a window size 10 (Figure 5.5g and Figure 5.5h).

Without active error analysis, the estimated parameter would be taken from the local GA with a window size of 30. The large window size would minimize the variance of estimated parameter, and filtered out the transient information from other local GAs with smaller window sizes. As shown by Figure 5.5f and Figure 5.5g , there was no observable improvement on tracking the change in the system pole. This experiment illustrated that distributed fitness function's full potential would not be utilized if active error analysis was not applied on the extra information available.

Experiment #4: Combining the distributed fitness function with active error analysis

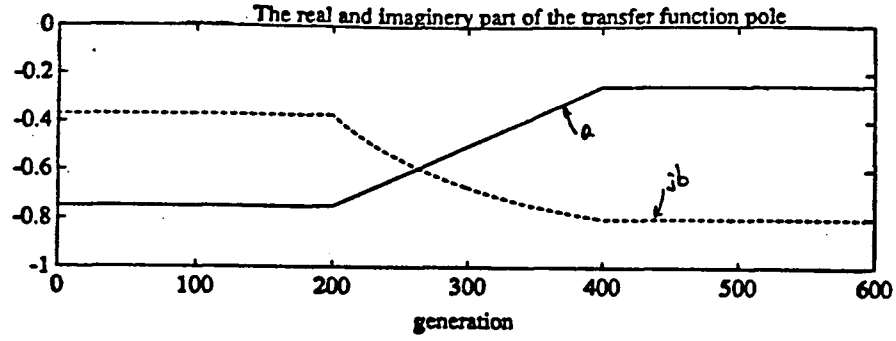
Figure 5.6a and Figure 5.6b show that the tracking performance of the PGA configuration #2 with

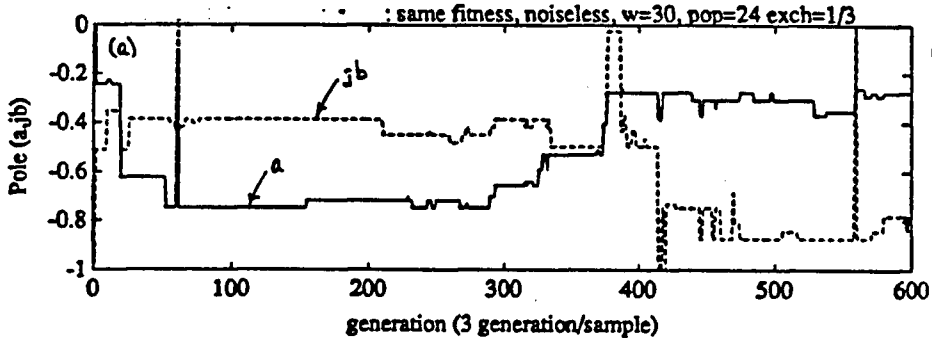
and without active error analysis in a noiseless environment. The active error analysis reduced the response delay from 100 generations to 50 generations. Active error analysis had used the transient information from the local GA of window size 10 to reduce the delay.

In a noisy environment, the active error analysis would be biasd more towards the local GA with large window size. In the present of noise and under steady state condition, estimated parameters variance from the local GA with a small window size would be quite large and the local GA with larger window size gave better estimation of system parameters. Comparing Figure 5.6c and Figure 5.6d , one would notice that both set of curves were very similar, confirming that under noisy conditions, active error analysis would be biased towards the local GA with window size 30. However, the active error analysis did not reject transient information from other GA with smaller window sizes. It still managed to reduce the response delay from 100 to 50 generations.

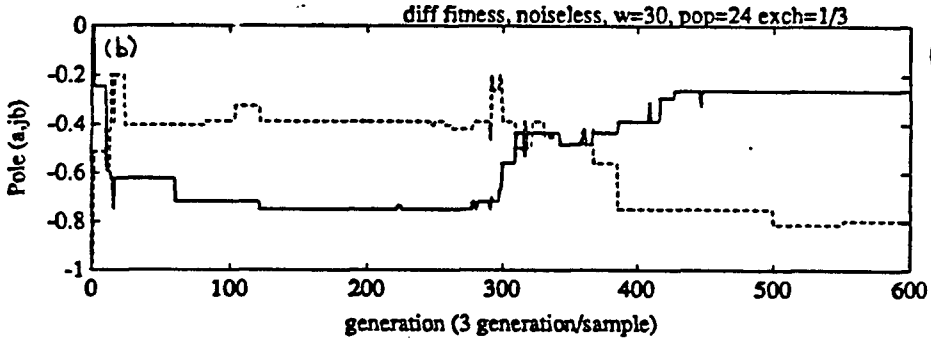
Experiment #5: Compare PGA and GA performance

The performances of GA and the PGA configuration #2 with the distributed fitness function and active error analysis were compared. The results are shown in Figure 5.7a to Figure 5.7d . The PGA configuration #2 actually outperformed GA in tracking parameter changes and had smaller variance in estimated parameters under both noiseless and noisy environment.

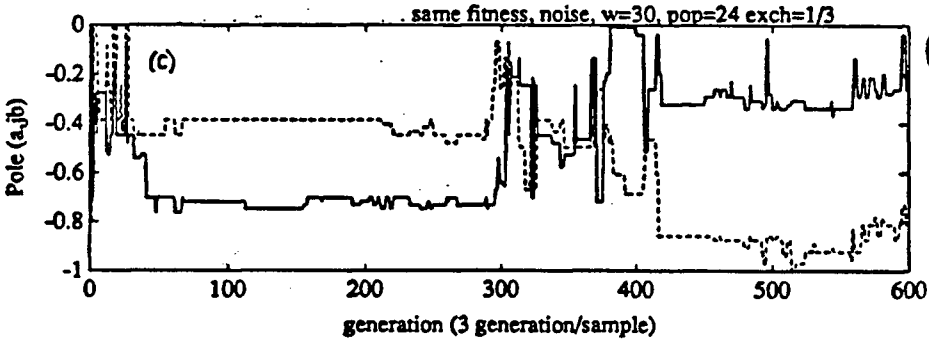




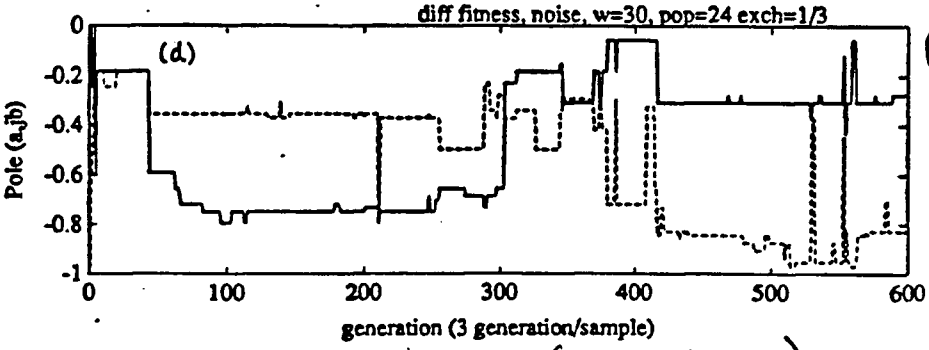
(a)



(b)

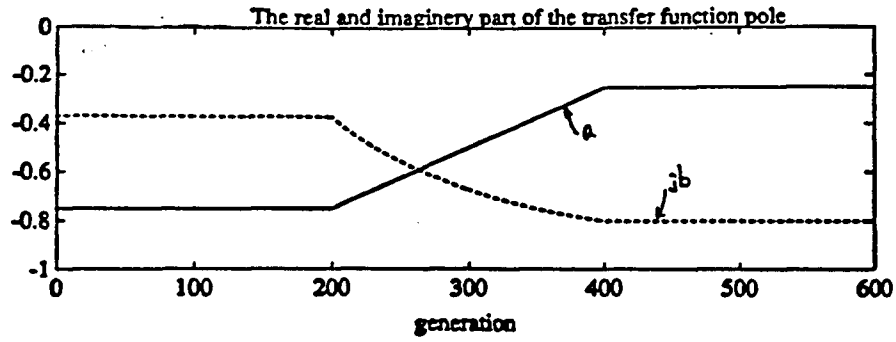


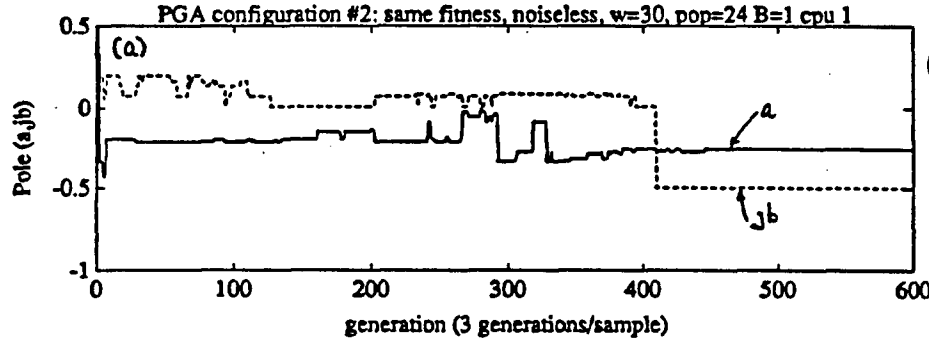
(c)



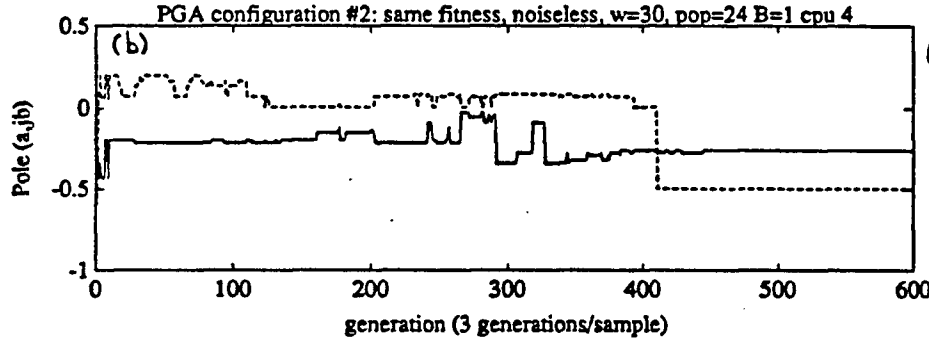
(d)

Fig 5.3 (Experiment #1)

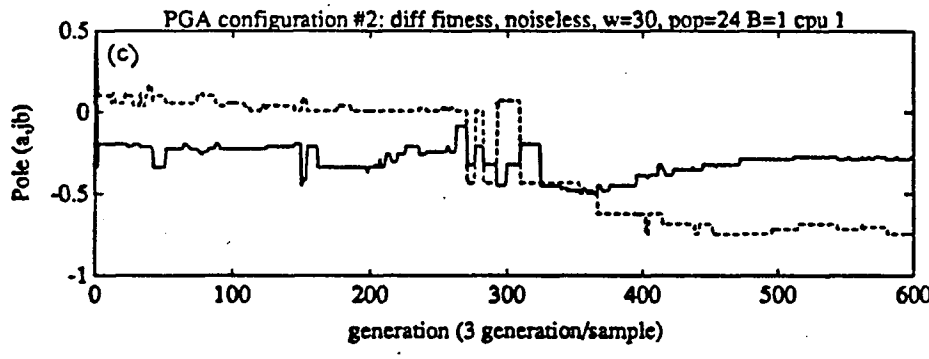




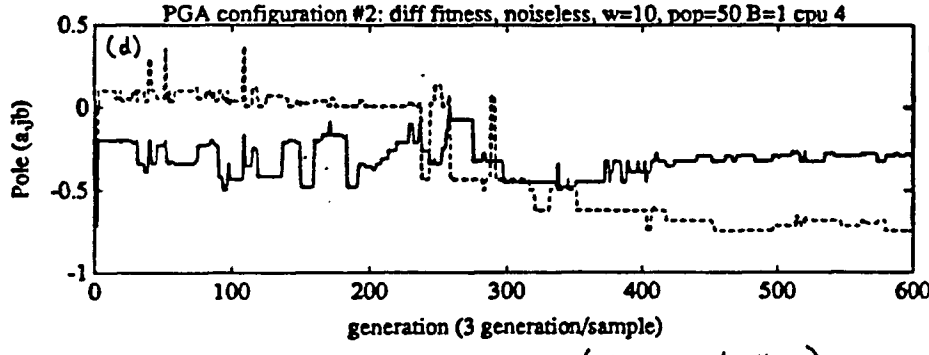
(a)



(b)

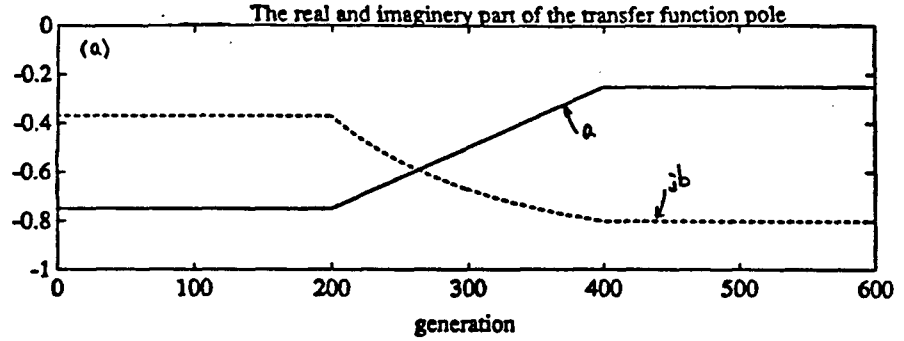


(c)

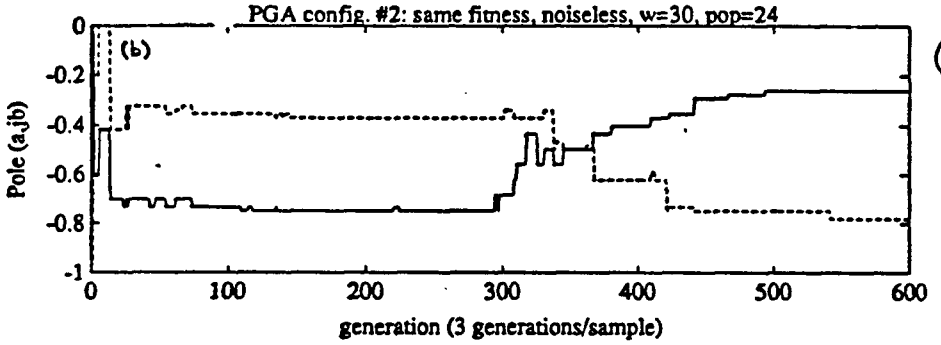


(d)

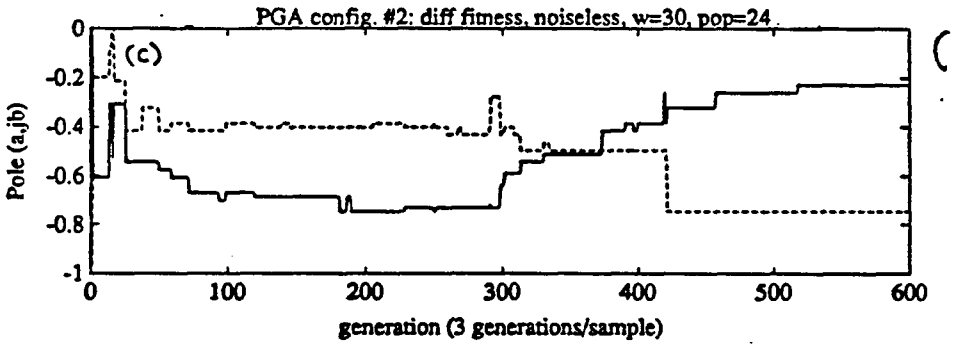
Figure 5.4 (Experiment #2)



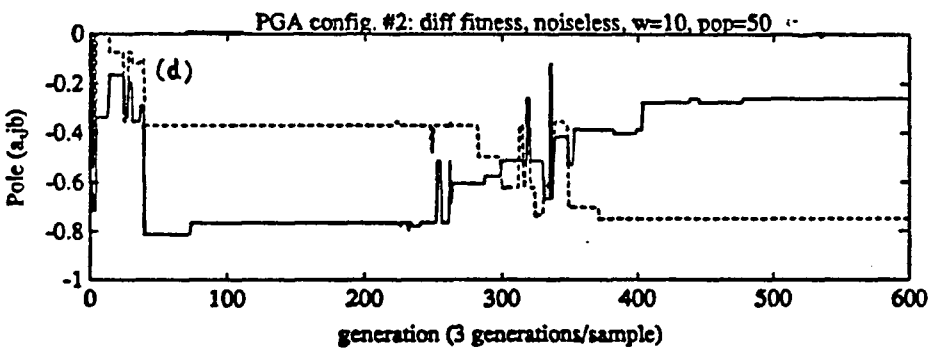
(a)



(b)

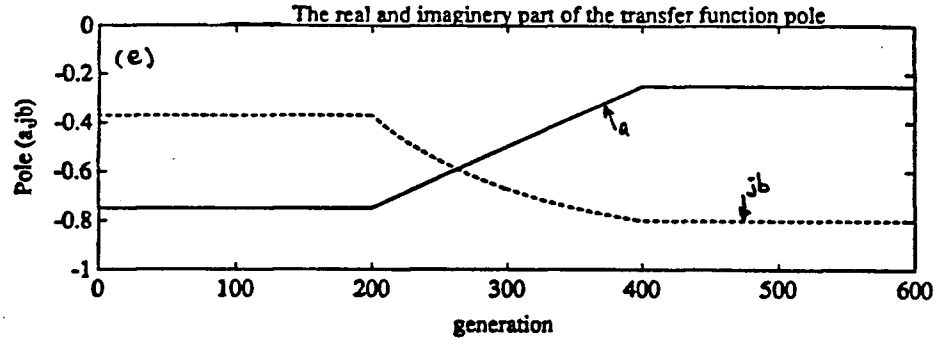


(c)

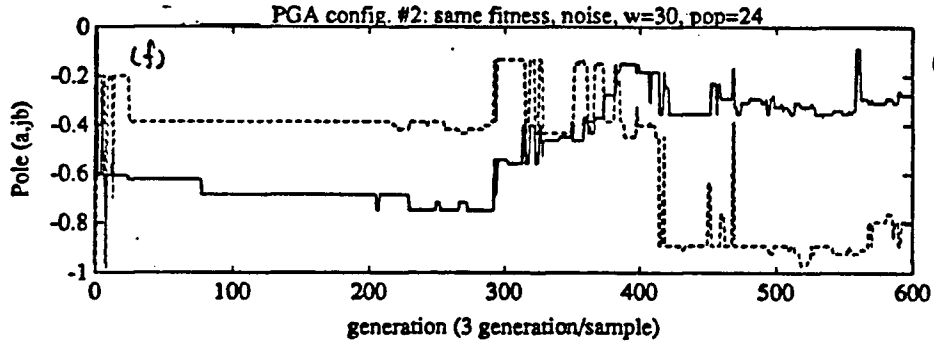


(d)

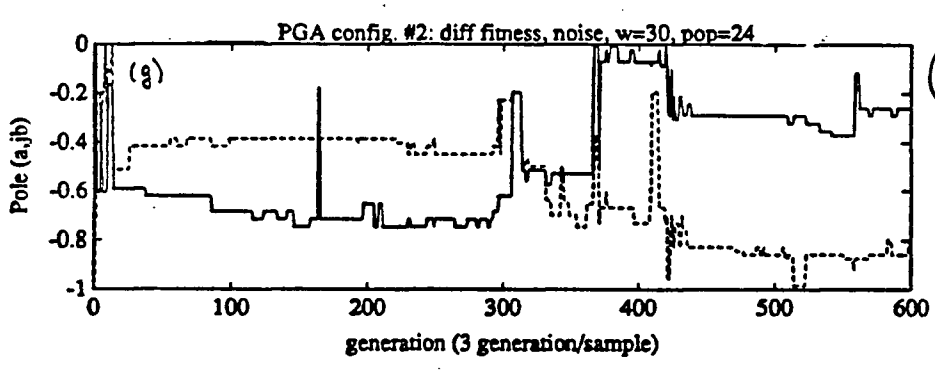
Figure 5.5 (Experiment #3)



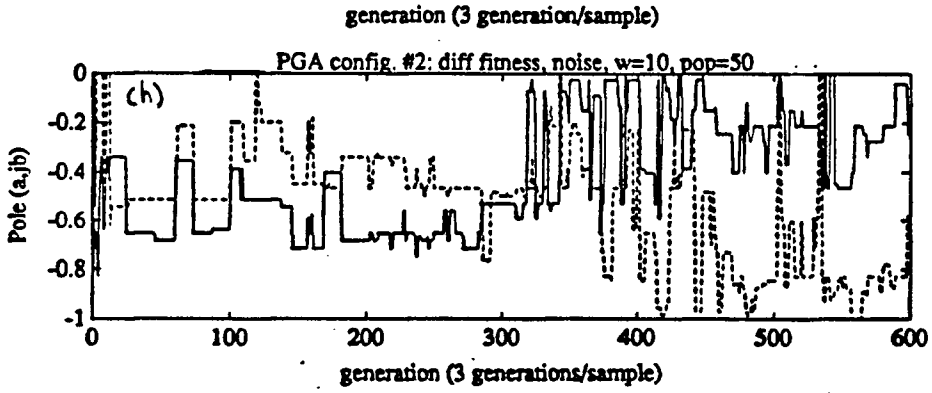
(e)



(f)

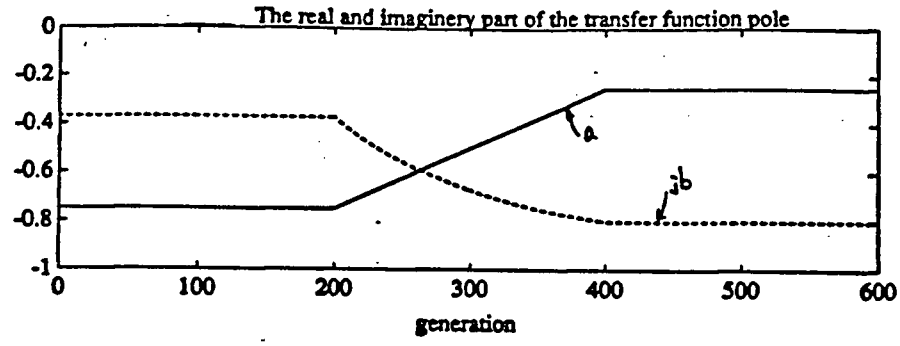


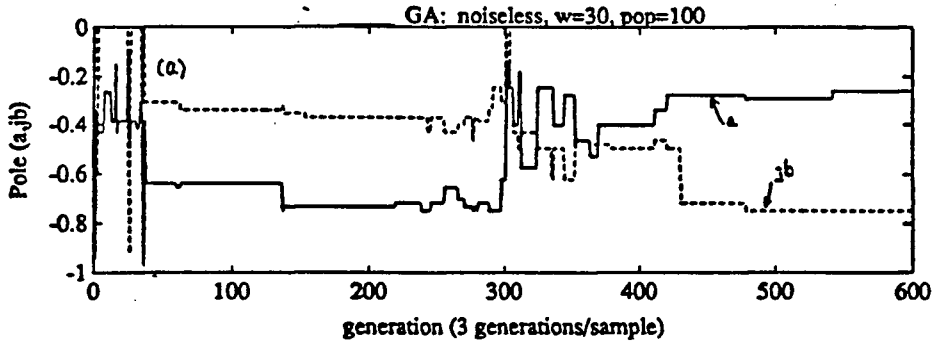
(g)



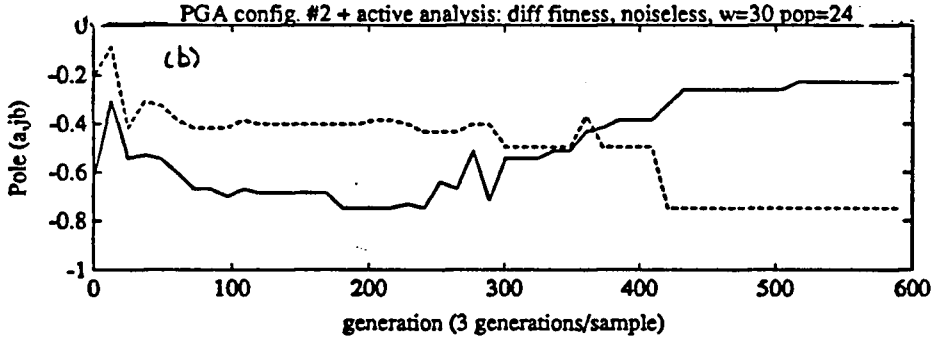
(h)

Figure 5.5 (Experiment #3)

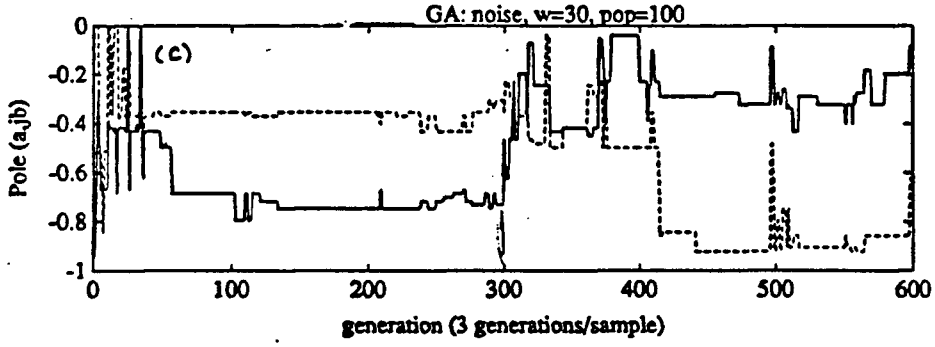




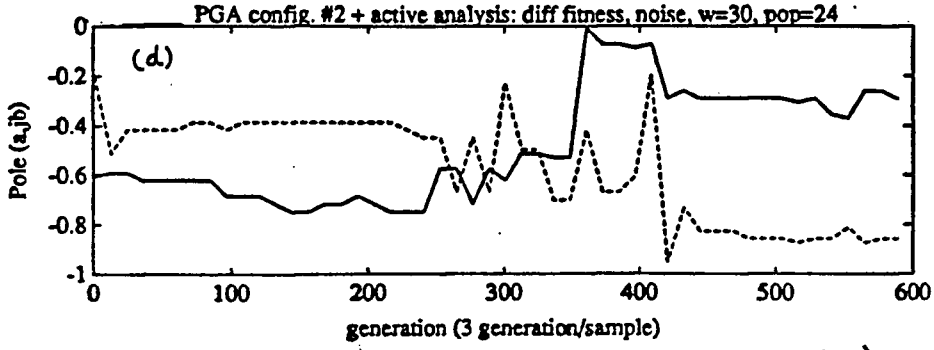
(a)



(b)

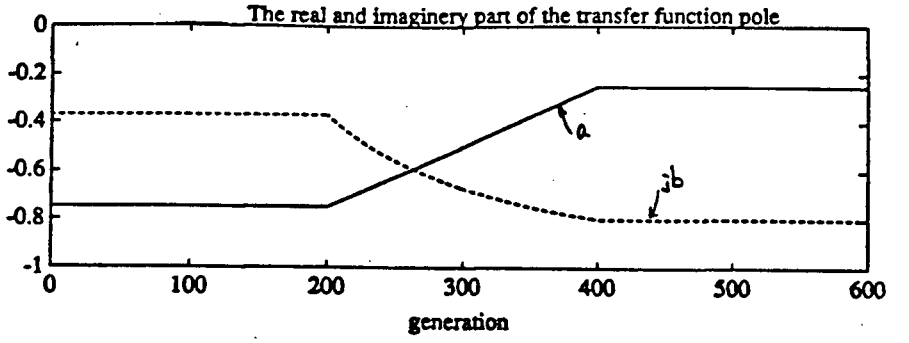


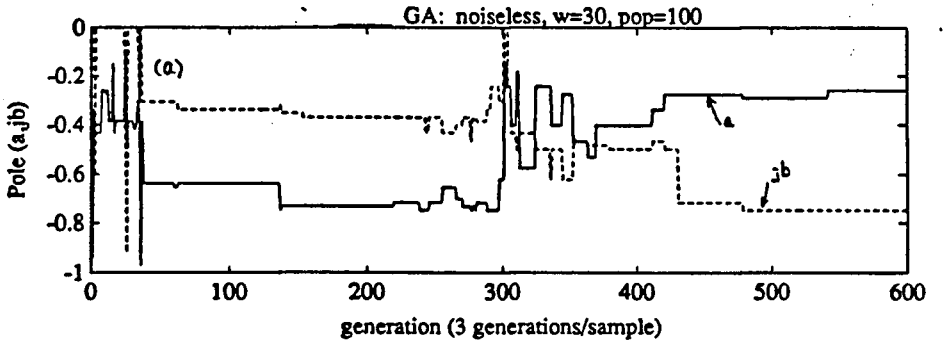
(c)



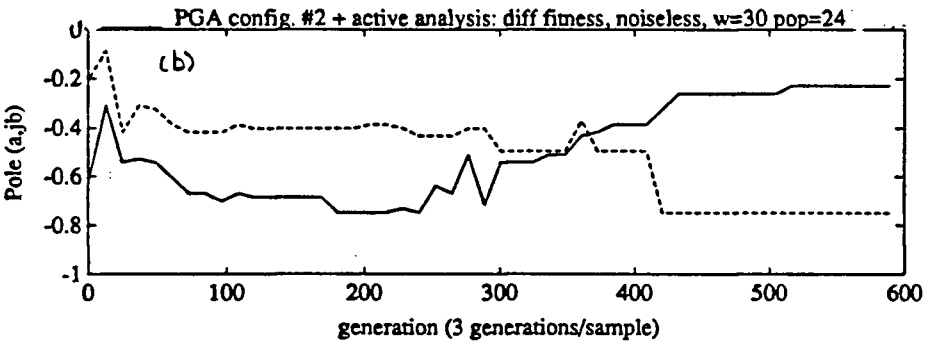
(d)

Figure 5.6 (Experiment #4)

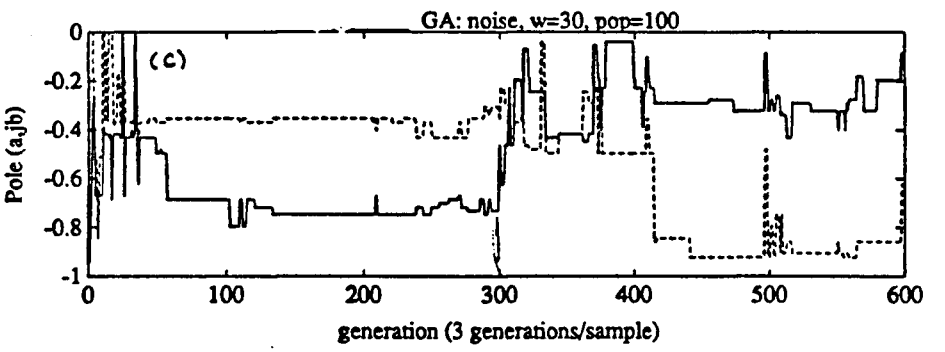




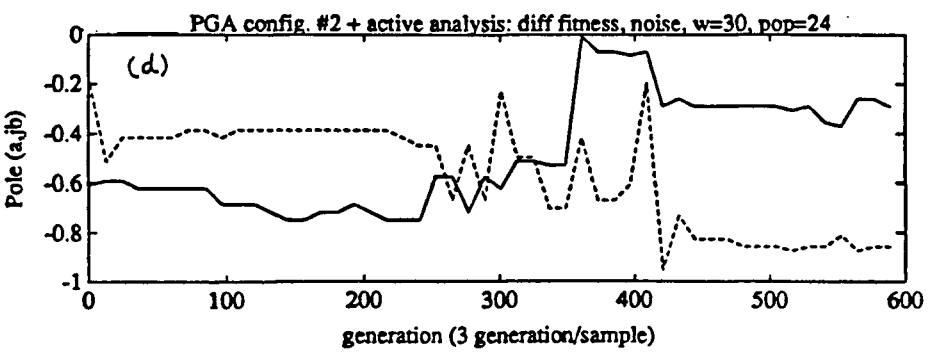
(a)



(b)



(c)



(d)

Figure 5.7 (Experiment # 5)

5.5 Comments and discussion

There are many advangages in using PGA (algorithm B) to gain speed over the traditional GA. Most of the advangages are mentioned in the beginning of the Chapter.

Almost all the PGAs implementation in the literature emulated the traditional GA by sequentially exchanging portions of the local populations among local or nodal GAs. Such exchange process is a passive mechanism since it may take several exchanges before a super individual be noticed by all the local populations.

In system identification (and control), the passive nature of the exchange mechanism of PGA can affect the parameter tracking rates of the algorithm. To improve PGA performance in system identification, three new mechanisms are used:

1. The distributed fitness function.
2. Broadcasting best individual among local GAs.
3. The active error analysis.

The distributed fitness function lets local GAs have different combination of window and local population size. It can significantly improve PGA parameter tracking rates and robustness.

Broadcasting removes the passive exchange mechanism so that all local GAs can interact with one and other simultaneously. Active error analysis is a method to compare best results from all local GAs and to choose the most appropriate output.

It is a reality that broadcasting may destroy the local population diversity. Fortunately, the distributed fitness function offers the local population effective protection against the disadvantages of broadcasting. The robustness and effectiveness of the distributed fitness function in protecting the local population diversity was tested. Experiments showed that PGA with distributed fitness function is much more robust than the conventional implmented PGA even when subjected to the highest broadcasting frequency of one per generation. Note that in the conventional PGA implementation, a sizable local population (>= 24) and a GA operator such as ranking[12] (to prevent a super-individual to reproduce too many and too quickly to dominate the local population) cannot protect local population diversity against broadcasting.

Experiment #3 and #4 show that the combination of the distributed fitness function, the broadcasting mechanism and the active error analysis produces noticeable improvement in estimated parameters tracking rate and estimated parameters variance. In experiment #5, PGA with these three new mechanisms actually outperformed the traditional GA in the key areas mentioned above.

A PGA with these three new mechanisms is a promising tool for many real-time, nonlinear system identification problems.

Chapter 6

Physical insight into how Genetic Algorithms work

The theory of schemata proposed by Holland

Holland (1975) has proposed the theory of schemata to explain some theoretical properties of Genetic Algorithms. A schema is a template describing a subset of strings with similarities over certain string positions. For example, given the following binary strings:

$$\begin{aligned} A &= 11100011 \\ B &= 11011011 \\ C &= 11011111 \end{aligned} \tag{6.1}$$

the template "11****11" is the common schema h for string A, B and C (" * " is the "don't care" state).

Holland introduced two important terms: the defining length of schema, δ(h) and the order of schema O(h). δ(h) is the length between a schema outermost defining positions. For example δ(11****11) = 8 – 1 = 7 and δ(10*11**1) = 8 – 1 = 7. The order of a schema is the number of defining positions for a string. For example O(11****11) is equal to four. The defining length δ(h) is a measure of the probability that a crossover may destroy such schemata. A schema with large δ(h) has a higher probability of being destroyed by a Crossover process than a smaller one. A small O(h) is better protected from Mutation than a larger O(h). Thus schemata with short defining length and low order stand the biggest chance of surviving to the next generation.

The combined effect of the three transition rules, Reproduction, Crossover, and Mutation can be estimated. The expected number of schemata h to survive into the next generation is the product of the

expected number from Reproduction and the survival probabilities of Crossover and Mutation [11].

$$m(h,t+1) = m(h,t)\frac{F(h,t)}{\bar{F}(t)}\left(1 - p_c\frac{\delta(h)}{l-1}\right)(1-p_m)^{O(h)}$$

$$\begin{aligned}
&where:\\
&m(h,t),\ m(h,t+1) = number\ of\ schema\ h\ at\ t,\ t+1\\
&F(h,t) = the\ average\ fitness\ of\ strings\ with\ schema\ h\ at\ t\\
&\bar{F}(t) = the\ average\ fitness\ of\ the\ population\ at\ t\\
&\frac{F(h,t)}{\bar{F}(t)} = the\ Reproduction\ factor\\
&(1-p_m)^{O(h)} = the\ probability\ schema\ h\ survives\ Mutation\\
&\left(1 - p_c\frac{\delta(h)}{l-1}\right) = the\ probability\ schema\ h\ survives\ Crossover
\end{aligned} \tag{6.2}$$

Note that the m(h,t+1) is a conservative estimate of schema h at t+1 since it assumes no Crossover among strings with similar schema. Holland's schemata theory states that the algorithm is going to converge towards the best, but there is no guarantee that it is converging to the optimum.

While Holland's schemata theory is helpful for predicting the searching power of GAs, it does not address many problems encountered in using GAs. Such as:

1. how to encode the parameter space to help the GAs searching process
2. how to avoid premature convergence
3. how to choose a realistic population size
4. how to modify the simple GA so that it becomes a less stochastic and more determinstic algorithm **(for certain applications only)**

We attempt to give a physical picture of how GAs work. With better understanding, it is easier to predict and even improve the performance of GAs in certain applications.

6.1 Genetic Algorithms search space

To understand the searching power and efficiency of GAs, one must visualize the search space of GAs. In everyday life, most of us work with real or integer number system. Functions such as F(x), F(x, y), F(x, y, z) are defined in a real number system as:

1. F(x) is a function defined in an one dimensional space spanned by the x unit basis
2. F(x, y) is a function defined in a two dimensional space spanned by the x and y unit basis.
3. F(x, y, z) is a function defined in a three dimensional space spanned by the x, y, and z unit basis.

The real number and the integer systems are easy for humans to understand. However, the binary number system is the key to unlock and to understand the searching power of GAs.

6.1.1 The binary unit basis

For a given resolution, it is possible to convert any real numbers x, y, z into binary notations B(x), B(y), B(z) of l bits long. Instead of associating B(x) as a decimal number,

$$B(x) = a_1 a_2 .. a_l \quad \equiv \quad x = a_1 \times 2^{l-1} + a_2 \times 2^{l-2} + \ ... \ + \ a_l \tag{6.3}$$

B(x) can be treated as the summation of l binary unit basis.

$$B(x) = a_1 a_2 ... a_l = a_1 V_1 + a_2 V_2 + \ ... \ + a_l V_l$$

$$where:$$

$$a_1, \ a_2 \ ... \ a_l = 1 \ or \ 0$$

$$V_1 = 00...001 \tag{6.4}$$

$$V_2 = 00...010$$

$$V_3 = 00...100$$

$$...$$

$$V_l = 10...000$$

Where $V_1, V_2, V_3, \ ... \ V_l$ are the binary unit basis for any binary notation of l bits long, and $a_1, a_2, a_3, \ ... \ a_l$ are the associated binary unit values.

Since any real (at a given resolution) or integer number can be converted to an unique binary notation, the binary unit bases, $V_1, V_2, V_3, \ ... \ V_l$ are the fundamental unit basis vectors that are similar to any real axis unit basis. A hypercube spanned by the binary unit bases $V_1, V_2, V_3, \ ... \ V_l$ (detailed description of the hypercube in later section). Given a function F(x) defined in 0 <

$x < x_{max}$ (assuming x is an integer), the corresponding function in the binary domain will be $F(a_1,a_2,...a_l,\ V_1,V_2,...V_l)$ with $l = Rounded\ Integer(log_2(x_{max}))$. $F(a_1,a_2,...a_l,\ V_1,V_2,...V_l)$ is a function defined in a l dimension space spanned by the binary unit basis $V_1,\ V_2,\ V_3,\ ...\ V_l$

Finding the maximum or minimum of a function (optimization problem)

If a function F(x) is well behaved (continuous and monotonic) in the real number domain of interest, then many conventional gradient based methods can be used to locate the maximum or minimum of such a function. However if the function is poorly defined (discontinuous, multimodal or highly nonlinear), most conventional methods will perform poorly since such methods depend on a well defined function gradient.

Figure 6.1 shows that a function F(x) can be linear or highly nonlinear, continuous or discontinuous in different regions of x . If one wants to locate the maximum or minimum of F(x), either all the values of F(x) over the interested domain are examined sequentially (analytical form of F may not exist) or a statistical method such as Monte Carlo is used to give rough estimation of the optimum location (multiple simultaneous searches are possible but they are very difficult to coordinate).

However, if the real number (at a given resolution) domain x is mapped into the binary domain spanned by $V_1,\ V_2,\ V_3,\ ...\ V_l$, then a hypercube is formed. Figure 6.1 shows the mapping from a linear one dimensional variable x to a hypercube. The advantage of such binary mapping is that with a l dimensional space, there can be at least l possible **coordinated** search directions for the locations of F(x)'s maxima and minima. It turns out that GAs operators Reproduction, Crossover, Mutation are tools that designed for **coordinated** searches in a hypercube space.

GAs search space and the hypercube

In GAs, the parameter space is usually encoded in a binary string format. Letting l be the length of the binary string, the parameters search space of a GA will contain 2^l (x_{max}) different combinations. Instead of working on the real number system, GAs map the search space into a hypercube of dimension l (refer to Chapter 1 "Introduction").

Since a GA search space is mapped into a hypercube of dimension l, many interesting properties of the hypercube can be used to give some insights into how a GA works. Figure 6.2 shows a four

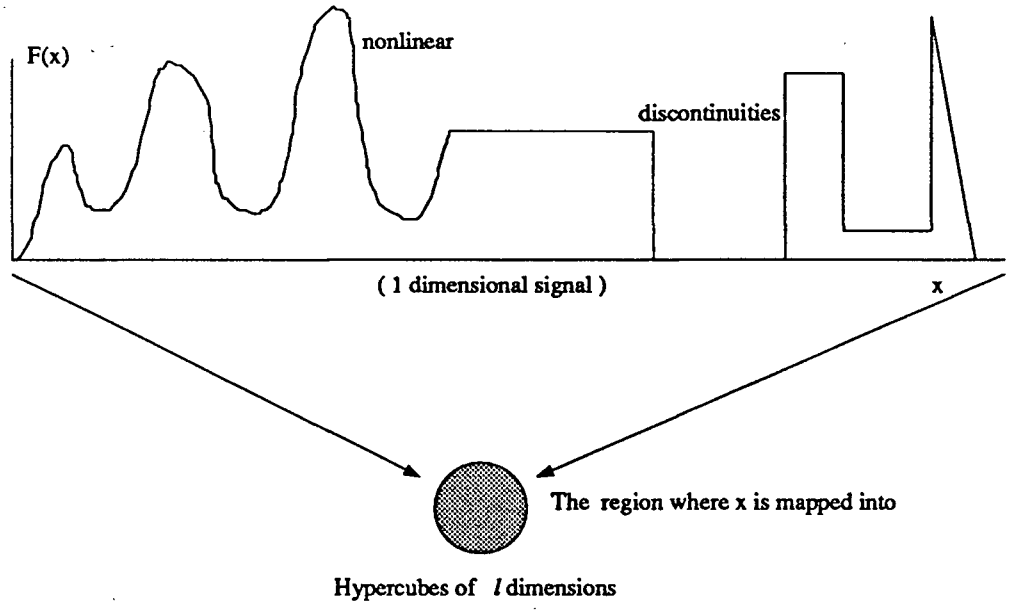


Figure 6.1: Mapping an one dimensional signal into a hypercube

dimensional hypercube with its vertices and links. Some of the important properties of a hypercube of dimension l are:

1. The search space of a GA is within the hypercube.
2. The optimum solutions sought by the GA are vertices of the hypercube.
3. The number of bit difference between any two different vertices is

   $0 < \quad the\ number\ of\ bit\ difference \quad \leq l.$
4. The minimum number of links between any two vertices on the hypercube is equal to the number of binary bit difference between them.

   For example, in Figure 6.2, going from vertex #12 (1100) to vertex #5 (0101) requires the spanning of at least 2 links. Thus there are 2 bits which are different between these two vertices.
5. Given any vertex on the hypercube of dimension l, the number of neighbours with n binary bit difference are shown in Table 6.1:

number of bit difference	number of neighbours
0	0
1	l
2	$l!$ / (2! (l-2)!)
3	$l!$ / (3! (l-3)!)
. . .	. . .
l - 1	l
l	1

Table 6.1 Number of neighbours vs number of bit difference

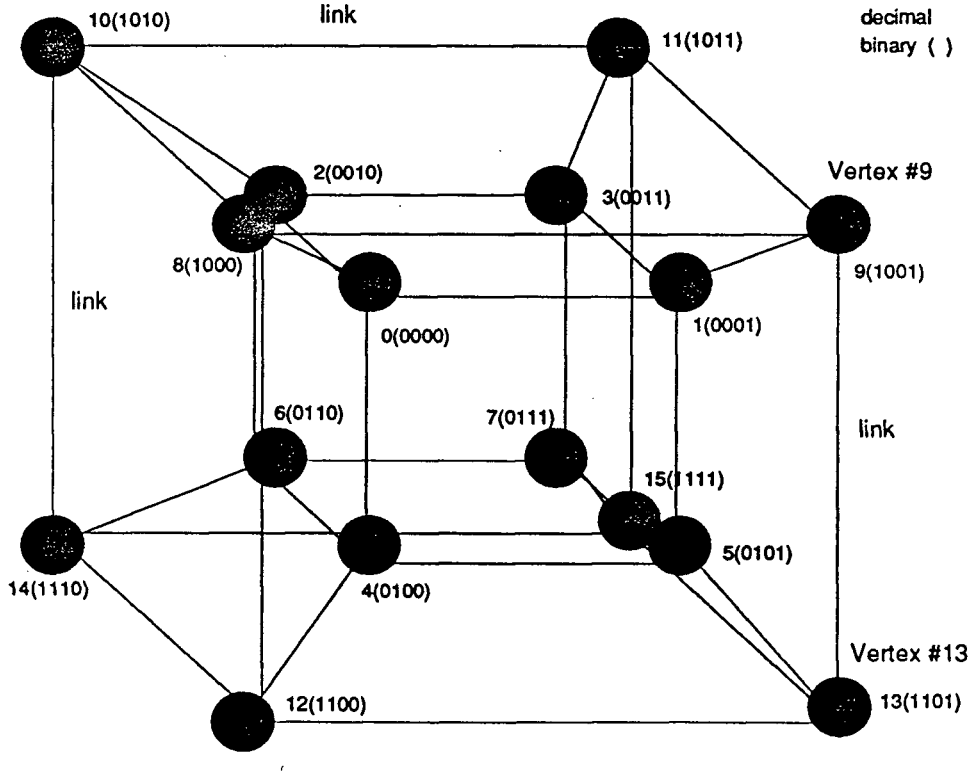


Figure 6.2: A four dimensional hypercube

6.1.2 The Reproduction, Crossover and Mutation operators on the hypercube

The Reproduction operator ensures that vertices (individuals) on the hypercube with good fitness will have many identical clones while vertices with poor fitness will be eliminated. The Reproduction

Number of bit difference	Location of the offsprings
1	(no new location)
2	On a plane (4 vertices) which contains the original parents
3	On a cube (8 vertices) which contains the original parents
4	On a hypercube of dimension 4 (16 vertices) which contains the original parents
n	On a hypercube of dimension n which contains the original parents.

Table 6.2 Locations of offspring

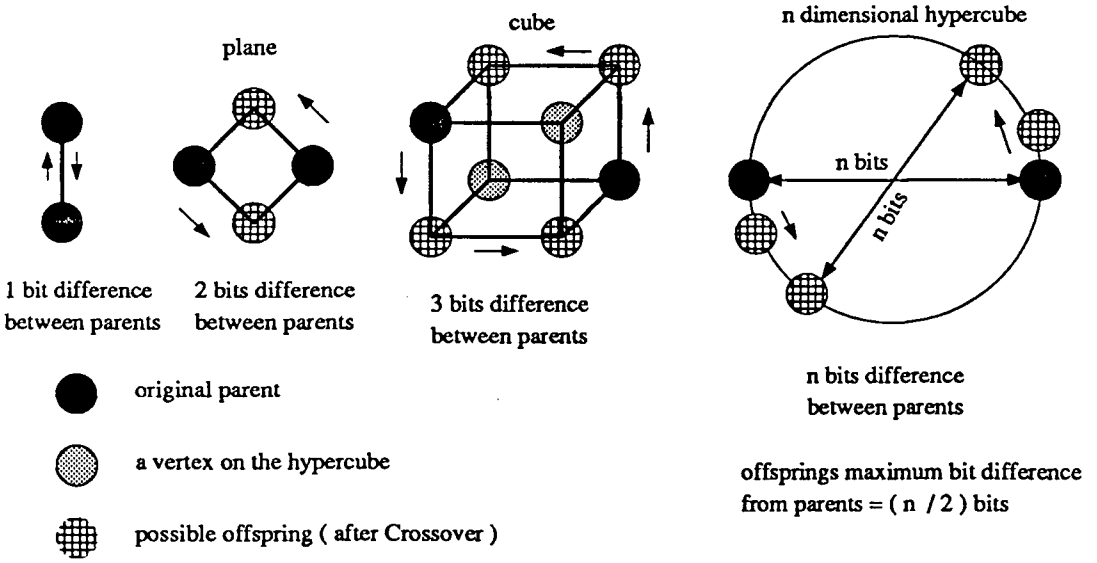


Figure 6.3: The possible locations of the offspring as a function of the bit difference between parents

The Crossover operator generates two new vertices (offsprings) that are in the proximity of the parents as shown in Figure 6.3. Thus the Crossover operator uses the bit difference information of the parents to generate offsprings such that the space spanned by the bit difference between the parents is searched.

operator is responsible for identifying relevant regions (many identical clones will be located in such region) in which the Crossover operator can perform intensive multiple direction searchs.

The Crossover operator is the most difficult operator to understand. It takes any two individual strings (vertices) and randomly selects a crossover point (the crossover point is randomly selected in a simple GA) interval $1 < k < l\text{-}1$ (l = length of binary encoded strings). Two new strings are then created by changing all bits between position k and l inclusively. For example:

$$\begin{aligned} &Before: \\ &A_0 = a_1 a_2 a_3 ... a_{l-1} a_l \\ &B_o = b_1 b_2 b_3 ... b_{l-1} b_l \\ &After\ (k=2): \\ &A_1 = a_1 b_2 b_3 ... b_{l-1} b_l \\ &B_1 = b_1 a_2 a_3 ... a_{l-1} a_l \end{aligned} \tag{6.5}$$

The Crossover operator can be more easily visualized much better on the hypercube space. Let

$$\begin{aligned} A_0 &= a_1 a_2 a_3 ... a_{l-1} a_l \\ B_0 &= b_1 b_2 b_3 ... b_{l-1} b_l \end{aligned} \tag{6.6}$$

be two vertices on a hypercube with dimension equal to l, using the concept of bit difference between vertices on the hypercube, the following statements can be made:

1. The bit difference between A_0 and B_0 determines the location of the offsprings A_1 and B_1 on the hypercube (Table 6.2 and Figure 6.3) .
2. If there is n bit difference between strings A_0 and B_0, the bit difference between offsprings A_1 and B_1 must also be n bits.
3. Depending on the value of k (the crossover point), offspring A_1 and B_1 can have 0, 1, 2, ... , integer (n/2) bit difference from A_0, B_0 respectively (n is the bit difference between A_0 and B_0 .

The Crossover process can also be explained in term of the binary basis $V_1, V_2, V_3, \ldots V_l$:

1. Given two vertices A_0 and B_0 on a hypercube of dimension l

$$\begin{aligned} A_0 &= a_1 a_2 a_3 \ldots a_{l-1} a_l \\ B_0 &= \bar{a}_1 a_2 \bar{a}_3 \ldots \bar{a}_{l-1} a_l \\ where: \ \bar{a}_1 &= complement\ of\ a_1 (1,0) \end{aligned} \tag{6.7}$$

2. A_0 and B_0 can be rewritten as

$$\begin{aligned} A_0 &= a_1 V_1 + a_2 V_2 + a_3 V_3 + \ \ldots \ + a_{l-1} V_{l-1} + a_l V_l \\ B_0 &= \bar{a}_1 V_1 + a_2 V_2 + \bar{a}_3 V_3 + \ \ldots \ + \bar{a}_{l-1} V_{l-1} + a_l V_l \end{aligned} \tag{6.8}$$

Let A_1 and B_1 be the products of Crossover process of A_0 and B_0, then

$$\begin{aligned} A_1 &= a_1 V_1 + a_2 V_2 + \bar{a}_3 V_3 + \ \ldots \ + \bar{a}_{l-1} V_{l-1} + a_l V_l \\ B_1 &= \bar{a}_1 V_1 + a_2 V_2 + a_3 V_3 + \ \ldots \ + a_{l-1} V_{l-1} + a_l V_l \\ with\ k &= 3\ (\ the\ crossover\ point\ ) \end{aligned} \tag{6.9}$$

3. Then

$$A_0 - B_0 = (a_1 - \bar{a}_1) V_1 + (a_3 - \bar{a}_3) V_3 + \ \ldots + (a_{l-1} - \bar{a}_{l-1}) V_{l-1} \tag{6.10}$$

$$\begin{aligned} A_1 - B_1 &= (a_1 - \bar{a}_1) V_1 + (\bar{a}_3 - a_3) V_3 + \ \ldots + (\bar{a}_{l-1} - a_{l-1}) V_{l-1} \\ &= (a_1 - \bar{a}_1) V_1 + (a_3 - \bar{a}_3) \bar{V}_3 + \ \ldots + (a_{l-1} - \bar{a}_{l-1}) \bar{V}_{l-1} \end{aligned} \tag{6.11}$$

$$where: \ \bar{V}_k = -V_k$$

Since the binary basis are the same for

$$A_o - B_o \ \ and \ A_1 - B_1 \tag{6.12}$$

implies that the bit difference between A_1 and B_1 must be the same as that of A_0 and B_0.

4. Crossover is similar to exchanging binary basis. For example, the new offspring A_1 loses the binary vectors $V_3, \ldots V_{l-1}$ (from Crossover) and B_1 gains the binary vector $V_3, \ldots, V_{l+1}$ (from the Crossover). Thus the direction A_1 moves from A_0 should be exactly opposite to the direction B_1 moves from B_0 within the smaller hypercube defined by the bit difference between the parents.

While choosing two parents is a random process (in the conventional GAs), Crossover is a structure searching process in the hypercube space (Even though the Crossover point k is randomly chosen, the offsprings will still be inside the hypercube space spanned by the bit difference between the parents).

Since each binary basis signifies a move in a certain direction in the hypercube, exchanging binary basis in crossover is actually a way to use one, two, or all different binary basis between the two parents to find new vertices in the space spanned by all the different binary basis between the two parents. For example, given the following strings:

$$A_0 = a_1 a_2 a_3 a_4 a_5 a_6$$
$$B_0 = a_1 \bar{a}_2 \bar{a}_3 \bar{a}_4 \bar{a}_5 a_6 \tag{6.13}$$

1. If the crossover point k is 2, the offsprings A_1 will move $\bar{a}_2\bar{a}_3\bar{a}_4\bar{a}_5$ away from A_0 and B_1 will move $a_2a_3a_4a_5$ away from B_0. Thus no new offsprings are created.
2. If the crossover point k is 3, then A_1 will move $\bar{a}_3\bar{a}_4\bar{a}_5$ from A_0 and B_1 will move $a_2a_3a_4$ from B_0. Thus 3 different bits are tested.

The probability of applying the Mutation operator to any bit in a string is usually quite low. Mutation can be seen as a local search algorithm on the hypercube. Mutation operator allows any vertex (an individual) to move to any one of its closest neighbour randomly. This operator is a random process and is effective only when the solution is only a few bit differences from the estimated one.

Visualizing the combined effect of Reproduction, Crossover, and Mutation process

Figure 6.4 shows the combined effect of Reproduction, Crossover on a randomly generated initial population. In step #0, the randomly generated individuals fitness values are calculated. Individuals with poor fitness are eliminated and individuals with good fitness are rewarded with more duplicates (step #1). The Reproduction operator seeks out the important space of the hypercube by letting vertices with good fitness to have more duplicates.

Although Crossover is a stochastic operator (the parents are randomly chosen and the crossover point k is randomly generated), its outcome (offspring) is quite predictable. It is seeking a combination of bit difference from the two individual parents to generate offsprings that may be closer to the solutions space. Since there are more copies of individuals with good fitness, there could be several different mates for such individuals. Thus the hypercube space (or bit patterns) around such individuals is searched more intensively (step #2). The combination of the GA operators Reproduction and Crossover gives GA the searching power.

The offsprings of the 2nd generation (step #3) fitness are calculated. The process of Reproduction, Crossover is repeated. Mutation operator is a local searching technique (on the hypercube). It is effective when the estimated solution is only one or two bits from the optimum one.

The searching power of GA is demonstrated since space around good individuals are searched from multiple directions simultaneously with **coordination**. After only a few generations, one can see that more and more offspring will be much closer to the optimum solutions.

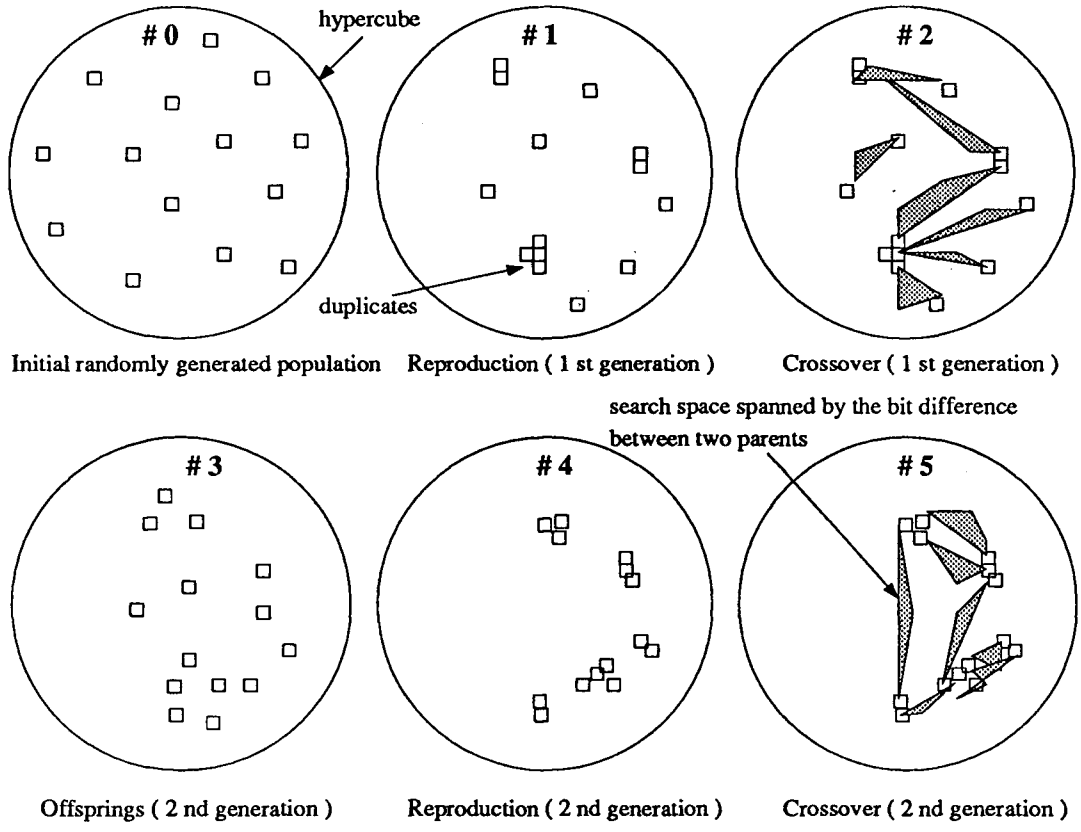


Figure 6.4: The combined effect of Reproduction, Crossover

6.2 The importance of encoding the parameter space properly

The effectiveness of GAs in searching toward the optimum value depends on how the parameters are encoded. Improper encoding of the parameters for GAs can increase the difficulties of the search similar to that of finding a needle in a haystack. For example, an impulse-like fitness function (Figure 6.5) would reduce GAs (or any other methods) to something no better than a blind search.

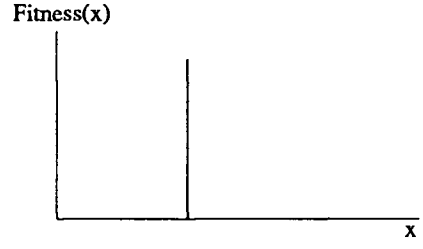
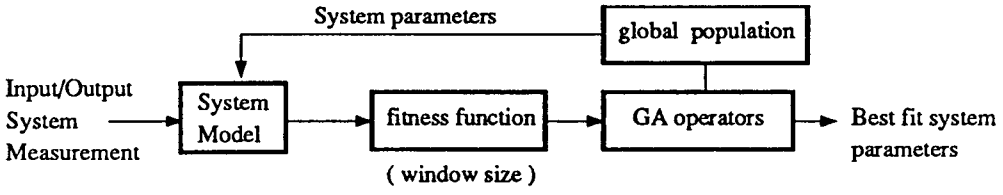
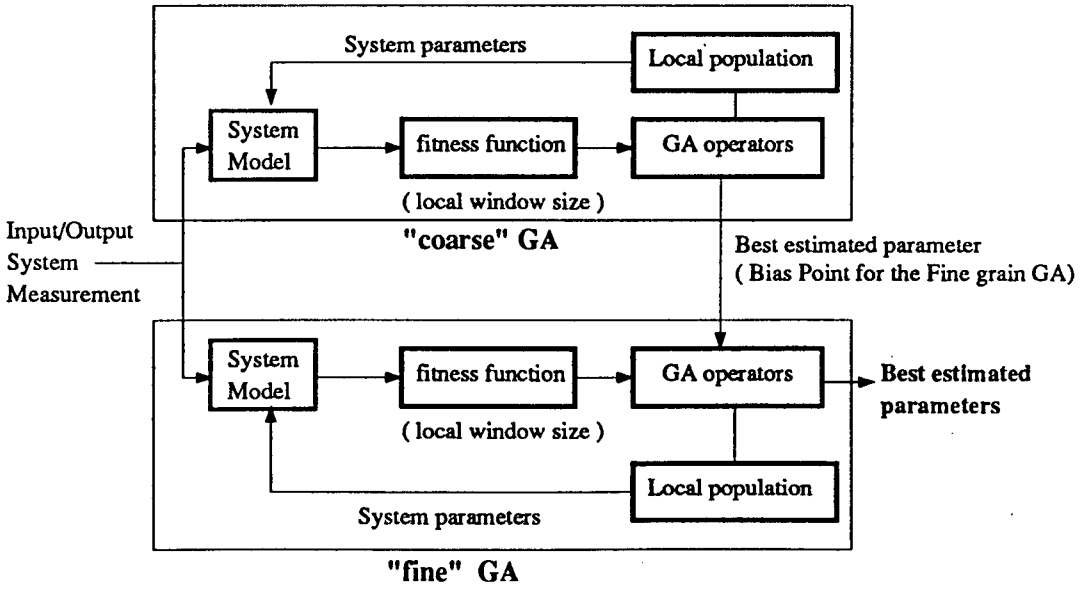


Figure 6.5: An impulse-like fitness function

In one of the experiments of system identification using GAs, we tried to reduce the size of the global population and increase the parameters convergence rate by shortening the individual binary string length. Instead of using the conventional GAs, a new combination GA is devised. Figure 6.6 shows how the conventional GA is divided into "coarse" GA and "fine" GA. The "coarse" GA will try to give a rough estimation of the system parameter (Figure 6.7, using a coarse grid space — less bit) to the "fine" GA. The "fine" GA will use the rough estimation of system parameters from "coarse" GA as a bias point to set a search space with much finer grid size to achieve the proper resolution. Since "coarse" GA is designed to give only a rough estimation of the parameter, its parameter convergence rate should be much higher than that of the conventional GA. Table 6.3 shows the number of bits used by the conventional, the "coarse" and the "fine" GA.



GA (Sequential Genetic Algorithm)



The new combination GA

Figure 6.6: Block diagram of conventional GA and new combination GA

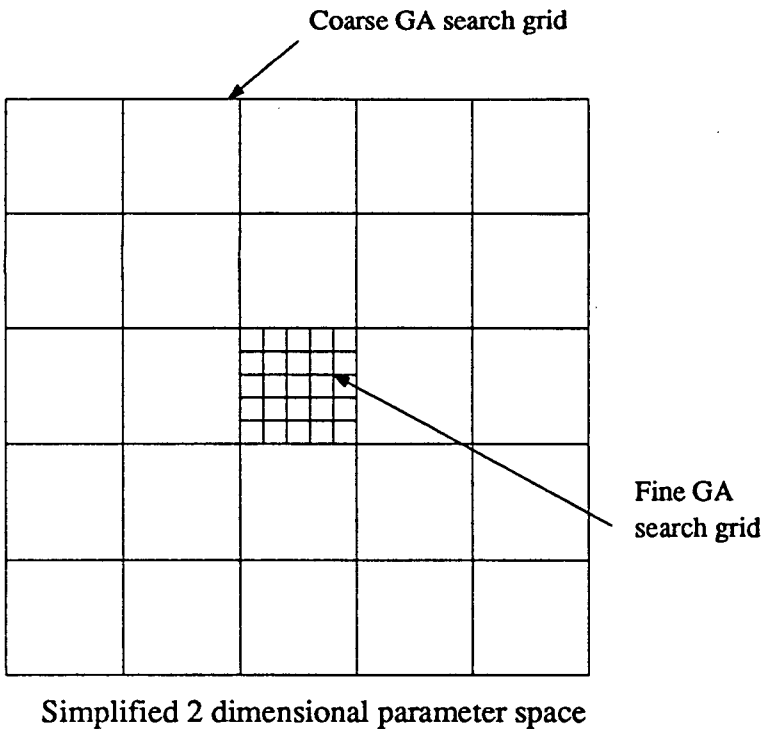


Figure 6.7: "Coarse" and "Fine" GA grid size

Parameters	Conventional GA (bits used)	"Coarse"GA (bits used)	"Fine" GA (bits used)
d (delay)	2	2	2
α	7	4 - 5	4 - 5
β	7	4 - 5	4 - 5
γ	7	4 - 5	4 - 5
δ	7	4 - 5	4 - 5
g	7	4 - 5	4 - 5
Length of individual binary string	37	22 - 27	22 - 27

Table 6.3 Number of bits used for the system parameters

Since the string length of the "coarse" and the "fine" GAs are less than the original one, then based on [8] theory, the population size of "coarse" and "fine" GA can be reduced significantly. Thus the execution time of the new combination GA could be much shorter than that of the conventional

GA. The system model, fitness function and setup in the experiment are identical to those described in Chaper 5. The experimental result of the new combination GA is shown in Table 6.4.

"Coarse" GA Population size	Length of individual binary string	Results
24	22	failed to converge
	27	good (converge rapidly)
30	22	failed to converge
	27	failed to converge
50	22	failed to converge
	27	slow convergence rate

Table 6.4 The performance of "coarse" GA with various size of population

Contrary to our expectations, the performance of the new combination GA was poor and unreliable. For individual string length of 22, the "coarse" GA failed to converge for all three different population sizes (4 bits are used to encode α, β, γ, δ, g). With string of 27 bits long (5 bits for each parameter), "coarse" GA is still unreliable.

In an attempt to reduce the string length for the "coarse" GA, the parameters were encoded incorrectly. By encoding the parameters into only four or five bits (Only the most significant bits of the parameters are used), all bits have equal importance. In the other word, any individual that does not match completely the solution bit by bit will have a poor fitness. From the hypercube model, the solution space for the "coarse" GA is reduced to a few locations on the hypercube. This is similar to having a impulse-like fitness function and a GA is no better than a random blind search.

If seven bits are used for each parameter, out of the 37 bits in an individual string, 15 bits (the lower order bits of the encoded parameters) are very similar to the "don't care" bits. Such individuals have fitness values very similar to the optimum ones. With 15 "don't care" bits, more than two or three thousands of such individuals will locate at many sections of the hypercube (see the neighbours of the hypercube in Table 6.1). For some applications (system identification) and

certain combination of l and D, (l = the dimension of the hypercube and D = the number of "don't care" bit, the probability of finding a good solution (not the optimum) can increase significantly.

6.3 Comments and discussion

While Holland's schemata theorem gives an excellent explanation to the searching power of GAs, it does not provide a good physical model of GA since the theorem is based on stochastic models.

We have shown that by mapping the search space of GAs into a hypercube, a working physical model of GAs can be built. With a physical model, GA mechanisms can be analyzed with more precision. Such a physical model may also allow one to further fine tune the GA mechanisms for better performance in a specific area.

The physical model gives a more precise explanation of GAs searching power, and it can help to resolve many other future technical issues such as the following:

A more realistic estimation of GA's population size

In the past, the schemata processing rate [11] (based on Holland schemata theorem) is used to estimate the optimum population size. The analysis [8] may be too conservative since it states that the population size of GAs should increase exponentially as string length is increased. Such analysis contradicts with concept of the bit difference geometry of the hypercube.

How to encode the parameter space

In the past, trial and error was used to find the optimum encoding for the parameters. The "coarse" and "fine" GA example illustrates how easily a mistake can be made in encoding. The hypercube physical model explained why "coarse" GA performed so unreliably. In certain applications, stuffing binary string with "don't care" bits can help GAs to find good solutions (vertices) on the hypercube easier. For example, given the following binary string:

$$a_1 a_2 a_3 a_4 \ldots a_{l-2} a_{l-1} a_l \tag{6.14}$$

it can be converted to

$$a_1 * a_2 * a_3 * a_4 \ldots a_{l-2} * a_{l-1} * a_l \tag{6.15}$$

The "don't care" bits " * " increase the bit range spanned by the number of vertices with fitness that are close to optimum, and thus increase the chance that GAs can find good solutions. Although the global search space increases exponential with number of bits in the binary string (2^l), the dimension of the search space increases only linearly (l).

Initial randomly generated population replaced by predefined populations

A different initial GAs population pattern can be used instead of a randomly generated one. Some prior knowledge of the search space, such as minimum space resolution, can be used to improve convergence rate. Such a technique can provide a better coverage of the search space than a randomly generated one.

Improving the Crossover operator

Choosing two individuals for Crossover should not be totally random. The hypercube physical model of the search space states that

1. Crossover between parents of 1 bit difference produces no new offsprings
2. Crossover between parents with bits difference in the "don't care" region produces no "new" offsprings.

If the above precautions are taken, the Crossover operator should become much less noisy (stochastic) and behaves more like a structure search.

Modified GAs are powerful multidimensional, almost deterministic (with some stochastic favors) optimization algorithms.

Chapter 7

Conclusions

The GA was used to estimate the compliance of the hydraulic system on the UBC teleoperated heavy duty excavator. Using real recorded data from the excavator, the GA successfully identified the compliance of the boom hydraulic circuit. The accuracy of the estimated compliance from GA was cross checked by a simulator. We found that the simulator output P_{BOi} agreed very well with measured P_{BOi} (Figure 3.8, within 10%), indicating that the estimated compliance C_{BOi} was accurate. GA was also used to estimate the compliances of multiple links simultaneously. The agreement between the simulation and estimated internal states of the hydraulic circuit (shows in Figure 3.9 to Figure 3.11) indicated that both the estimated compliances and the estimated internal states of the hydraulic system from GA were accurate (within 10%). The searching ability and power of GA were well demonstrated in this example.

Different algorithms and architectures were studied for a parallel implementation of GA. For the GA used in system identification, both population and fitness calculation could be parallelized. Such combination allows the use of the pipeline techniques, which improved the efficiency of the parallel system.

A PGA was implemented with sixteen T800 Transputers. It achieved a speedup factor of 12 over a traditional GA. This PGA was used to identify the compliance of the boom hydraulic circuit (Chapter 3). The estimated compliance from this PGA was almost identical to the one from traditional GA. With such a high speedup factor, real-time monitoring of hydraulic compliance and other hydraulic parameters is becoming possible.

While many papers on "PGA in function maximization" can be found in the literature, there seems to be no published paper on "PGA in system identification". This may be because of the inherent disadvantages in PGA, such as slower parameter convergence rate and lack of guarantee that it will converge to an optimum.

In Chapter 5, we addressed the different requirements between function maximization and system identification and introduced three new mechanisms: the distributed fitness function, the broadcasting

mechanism and the active error analysis that could be used to enhance the performance of PGA. Simulation results confirmed the effectiveness of these three mechanisms. A PGA which incorporates these three mechanisms can actually outperform GA in key areas such as variance of the estimated parameter and parameter tracking ability.

There are many conventional methods available for linear system identification. If the system is nonlinear, the identification problem is much more complex and difficult. The GA is not recommended for nonlinear system identification if other identification methods can be used. However there are many nonlinear systems where conventional identification methods perform poorly or fail, the GA may become the only alternative method available. Even through there is no guarantee that GA will converge to an optimum solution, from our experience with nonlinear systems identification using the GA, we find that the GA usually provides solutions that are close to the optimum ones.

In Chapter 6, we derived a physical model that explain the fundamental properties of GAs. The physical model (a hypercube) not only provide an excellent explanation of GA searching power, but also provide insights to GAs user ways to improve the performance of GAs in most applications.

The hypercube physical model also explains why the GA and PGA are so successful in locating solutions near the optimum one in nonlinear system identification. In a typical nonlinear system identification application, the parameters are encoded in a binary string as follow:

$$\begin{gathered} \alpha|\beta|\gamma|\delta \ \ldots|\lambda \\ a_1 a_2 a_3 .. a_n b_1 b_2 b_3 \ldots b_m \ldots \end{gathered} \tag{7.1}$$

Where n bit is used to encode α, m bit is used to encode β, ... etc. Usually only half of the bit used for each parameter is significant (the most significant bits of the parameters, eg. $a_1 a_2 a_3 a_4 a_5$). The insignificant or unimportant bits of the parameters can be classified as the "don't care" bits. Thus with so much "don't care" bits encoded in the binary string, the probability for the GA or PGA to find a solution in the hypercube space is quite high.

Finally, The fundamental properties of GAs are as followed:

1. By converting parameter search space from a real number system into a binary number system, GAs increase the dimension of the search space. Higher dimension search space allows multiple

search with different directions, thus increasing the efficiency of function maximization process in the search space by at least l times the linear method.

2. The hypercube configuration is an excellent tool to illustrate the operation of GA operators: Reproduction, Crossover, and Mutation. By eliminating individuals with poor fitness and cloning individuals with good fitness, the Reproduction operator can quickly locate areas of interest on the hypercube. On the hypercube, Crossover can be seen as a directed structural search method that uses the difference between the two parents to produce offspring in the direction (derived from the bit difference between parents) toward the optimum. Mutation is just a simple random neighborhood search around any vertex.

The physical model provides a new way to re-evaluate most of the important parameter settings of the GAs. From the last chapter, we point out that it is possible to derive:

1. A realistic population size that grows linearly instead of exponentially with respect to the binary string length (l = dimension of the hypercube) for certain applications.
2. The Crossover operator can be tuned so that it is behaves much more like a structural search and resembles much less as a stochastic operator.
3. Encoding the parameters properly so that premature convergence or blind search can be avoided.

With all the technical issues resolved, the **structured** GA is a powerful multidimensional, deterministic (with some stochastic flavor) optimization algorithm.

Appendix A The boom hydraulic circuit and its model

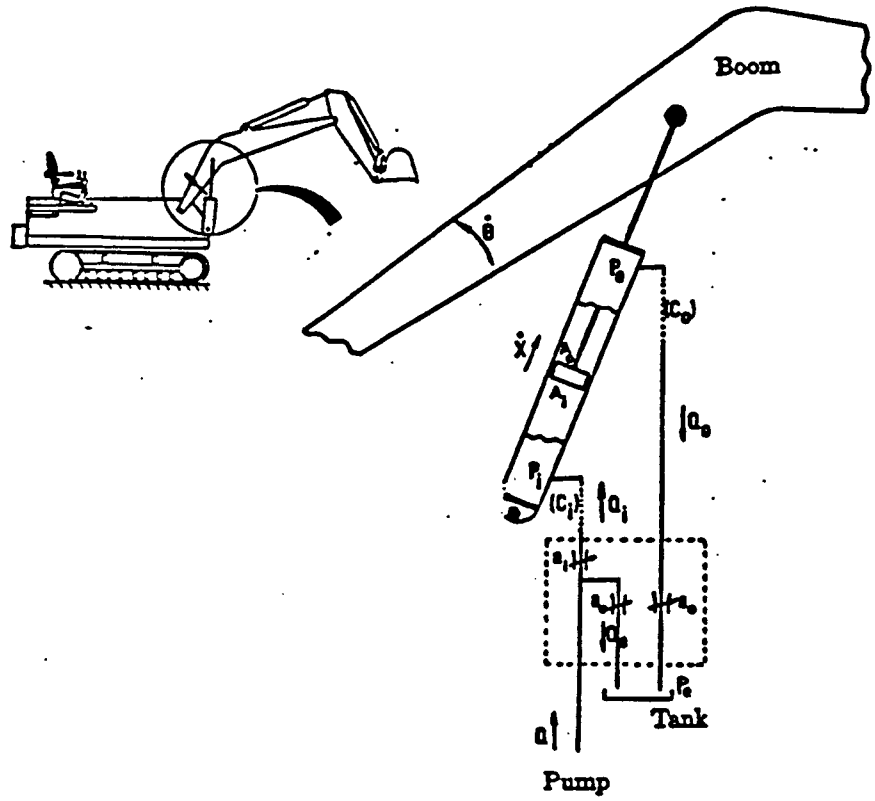


Figure A.1: The boom hydraulic circuit

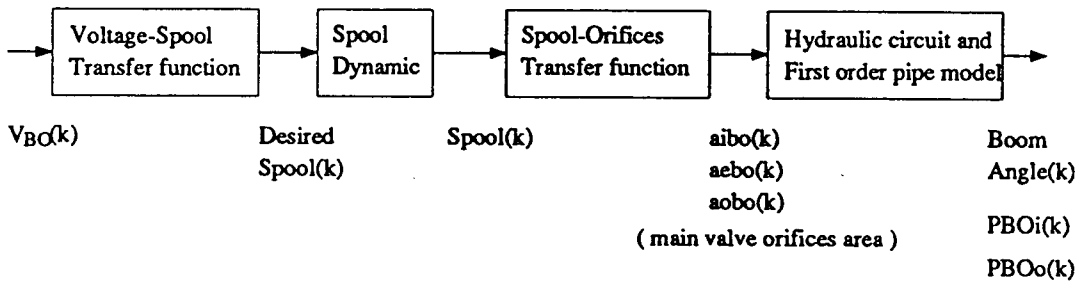


Figure A.2: The mathematical model of the boom hydraulic circuit

The Voltage-Spool Transfer function for the boom:

$$Desired\ Spool(k) = 0.0892V_{BO}^3 + 0.2352V_{BO}^2 + 0.2815V_{BO} + 0.1297 \tag{A.1}$$

The Spool dynamic:

$$\begin{aligned} &Discrete\ time\ model: \\ &Spool(k) = Desired\ Spool(k) \times \left(1 - e^{-a_0 T}\right) + e^{(-a_0 T)} \times Spool(k) \end{aligned} \tag{A.2}$$

The Spool-Orifices Transfer function:

HminSpool[1] = the dead band in the Spool = 0.188 inch.

a_{ebo}:

if (Spool <= HminSpool[1])

$$a_{ebo} = (-7.85\ in. \times Spool) + 1.5235\ in. \tag{A.3}$$

else if (Spool <= 0.376 in.)

$$a_{ebo} = -0.25\ in. \times (Spool - 0.376\ in.) \tag{A.4}$$

else a_{ebo} = 0.0 in.;

a_{ibo}

if (Spool <= HminSpool[1]) a_{ibo} = 0.0 ;

else if (Spool <= 0.376 in.)

$$a_{ibo} = 1.247\ in. \times (Spool - HminSpool[1]) \tag{A.5}$$

else

$$a_{ibo} = (3.927\ in. \times (Spool - 0.376)) + 0.2346\ in. \tag{A.6}$$

a_{obo}

if (Spool <= HminSpool[1]) a_{obo}=0.0 in.;

else if (Spool < 0.248 in.)

$$a_{obo} = (Spool - 0.248\ in.) \times HminSpool[1] + 0.0113\ in. \tag{A.7}$$

else if (Spool < 0.376 in.)

$$a_{obo} = (Spool - 0.376\ in.) \times 1.09\ in. + 0.1509\ in. \tag{A.8}$$

else

$$a_{obo} = (Spool - 0.376\ in.) \times 3.927\ in. + 0.1509\ in. \tag{A.9}$$

Appendix B Dynamic equations for hydraulic system in configuration a, branch #1

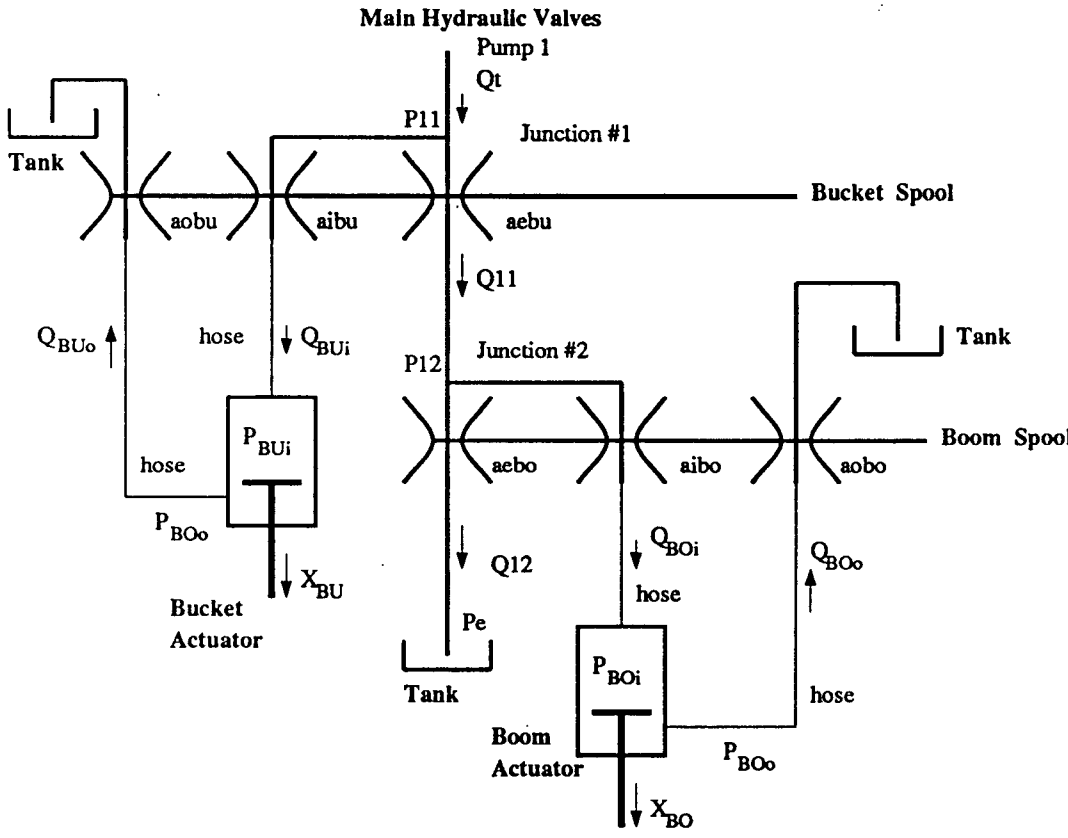


Figure B.1: The branch #1 of the main hydraulic valves in configuration a

B.1 Equations describing the hydraulic configuration a branch #1

Measured variables :

P_{BUi}, P_{BOi}, θ_{BU}, θ_{BO}, V_{BU}, V_{BO} (*or* $Spool_{BU}$ *and* $Spool_{BO}$)

Calculated from measured variables :

$\dot{P}_{BUi}$, $\dot{P}_{BOi}$, $\dot{X}_{BU}$, $\dot{X}_{BO}$, *the orifices areas* (B.1)

Estimated parameters : (pump side compliance)

C_{BUi}, C_{BOi}

Unknown internal states :

Q_{BUi}, Q_{BOi}, Q_{11}, Q_{12}, P_{11}, P_{12}

First order pipe model :

$$C_{BOi}\frac{\partial P_{BOi}(t)}{\partial t} = Q_{BOi} - A_{BOi}\dot{X}_{BO} \qquad (B.2)$$
$$C_{BUi}\frac{\partial P_{BUi}(t)}{\partial t} = Q_{BUi} - A_{BUi}\dot{X}_{BU}$$

Where

A_{BOi} = the boom actuator pump side piston area,

A_{BUi} = the actuator pump side piston area,

Equations for hydraulic flow :

$$Q_{BUi}(t) = k\, a_{ibu}(t)\sqrt{P_{11}(t) - P_{BUi}(t)}$$
$$Q_{11}(t) = k\, a_{ebu}(t)\sqrt{P_{11}(t) - P_{12}(t)} \qquad (B.3)$$
$$Q_{BOi}(t) = k\, a_{ibo}(t)\sqrt{P_{12}(t) - P_{BOi}(t)}$$
$$Q_{12}(t) = k\, a_{ebo}(t)\sqrt{P_{12}(t) - P_e}$$

Flow Constraint :

$$Q_t = Q_{11} + Q_{BUi} \qquad (B.4)$$
$$Q_{11} = Q_{12} + Q_{BOi}$$

B.2 Simplified orifices area calculation

The simplified hydraulic model does not contain the Voltage-Spool and the Spool-Orifices transfer functions. It assumes a step input and the orifices areas are described by the following equations:

The dynamics of the hydraulic valves :

$$
\begin{aligned}
a_{ibu}(t) &= a_{ibu1}\left(1 - exp^{-a_0 t}\right) \\
a_{ebu}(t) &= (a_{ebu0} - a_{ebu1})exp^{-a_0 t} + a_{ebu1} \\
a_{ibo}(t) &= a_{ibo1}\left(1 - exp^{-a_0 t}\right) \\
a_{ebo}(t) &= (a_{ebo0} - a_{ebo1})exp^{-a_0 t} + a_{ebo1}
\end{aligned}
\tag{B.5}
$$

Appendix C Speedup and efficiency definitions

Speedup is defined as:

$$Speedup = \frac{GA\ execution\ time\ for\ 1\ processor}{GA\ execution\ time\ for\ N\ processores}$$

where :

$GA\ execution\ time\ for\ 1\ processor = GA\ fitness\ calculation\ time(t_p) + GA\ operator\ time(t_s)$

$GA\ execution\ time\ for\ N\ processors = GA\ fitness\ calculation\ time(t_p/N) + GA\ operator\ time(t_s)$

$t_c = Communication\ overhead\ for\ N\ processor.$

Thus :

$$Speedup = \frac{t_p + t_s}{\frac{t_p}{N} + t_s + t_c} \approx \frac{t_p + t_s}{\frac{t_p}{N} + t_s} \quad if\ t_s >> t_c \tag{C.1}$$

Efficiency is defined as:

$$Efficency = \frac{Speedup\ obtained\ from\ N\ processsors}{N} = \frac{Speedup}{N} \tag{C.2}$$

Appendix D Overview of available computational resources

Details of the available hardware resources are shown in Figure D.1. The host computer is a Sun 3/280 68020 based CPU equipped with a terminal board, an Ethernet interface, and a disk storage subsystem. The distributed system consists of the following modules:

1. The Bus Masters (Parsytec-BBKV2): A transputer bus master is composed of an T800 transputer with 2 MBytes of dual-ported memory which is asynchronously accessible by the transputer and the VMEbus. The bus master also performs automatic conversion of data format, allowing compatible messages to be exchanged efficiently with the host computer.
2. Clusters (Parsytec-VMTM): Each cluster module consists of four Inmos T800 transputers each with 1 MByte of local memory, nine external serial ports, a 32x32 crossbar switch, and four link adapters for interfacing a transputer connection link to the VMEbus. The external serial ports and the crossbar can be used to connect a transputer from a cluster module to another transputer on any other cluster module. The 4 link adapters on the VMEbus provide the facilities to support broadcasting of data from the host computer or the Transputer bus master.

The Transputer T800 architecture:

The Transputer is designed and built on Hoare's concept "Communication Sequential Processing" [10]. All processes on the Transputers communicate synchronously through soft (internal channel) or hard (external transputer links) channels and therefore all concurrent processes synchronize automatically. The Transputer has a built in scheduler for all its processes. Any process that is waiting for input will be de-scheduled until its data become available. Thus the CPU does not spend any time idling. With high speed data links of (20 Mbit/s) and direct memory addressing to support the hard link features, there is very little interruption for the local CPU for communication among two Transputers. Such features make parallel processing on multiple Transputers feasible and efficient. The T800 Transputer hardware block diagram is shown on Figure D.2.

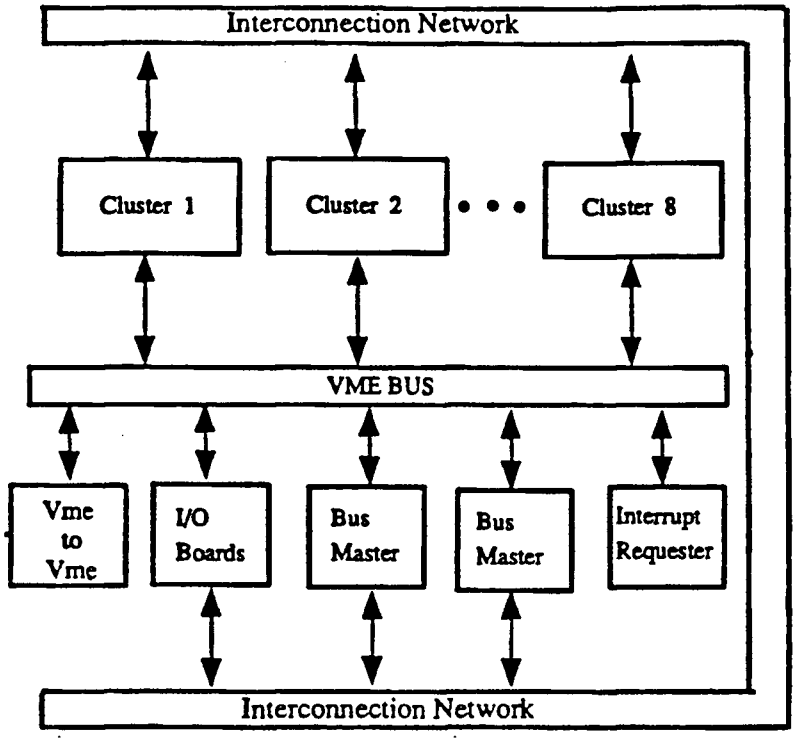


Figure D.1 Available hardware resources

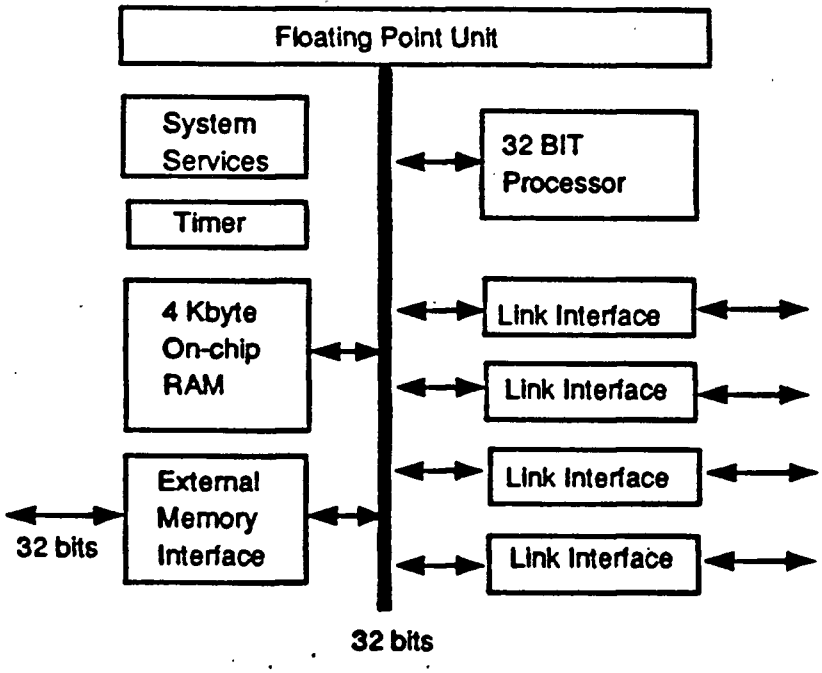


Figure D.2 The block diagram of the Transputer T800

Appendix E PGA implementation details

As discussed in Chapter 3 "Identification of hydraulic compliance with GA", to identify the single link hydraulic compliance, the global population size is set between 100–120. The ratio α is in the range between 8 to 16. The minimum local population size M_{loc} is set at 30 to protect the diversity of the local population (using the criteria discussed in next Chapter).

To achieve as much speedup as possible, algorithm A and algorithm B are combined. The maximum speedup is:

$$Max.\ speedup = \frac{120}{30} \times \frac{8+1}{2} \approx 16 \qquad \text{(E.1)}$$

Level 1: global population (algorithm B)

Given the minimum local population size of 30 and a global population of 120, there will be 4 local GAs. The 4 local GAs are connected in a ring network using the hard link of the Transputer.

Level 2: The Fitness Calculation (algorithm A)

With a local population size of 30, fitness calculation time t_{p2} and the sequential GA operators time (Reproduction, Crossover, Mutation) t_{s2} accounts for 80% and 20% of the execution time respectively. Such high ratio of α (t_{p2} / t_{s2}) fits the master- slaves scheme requirement. Each local GA unit will consist of one master and several slave processors. The master processor distributes individuals from the local population to slave processors. Slave processors calculate the fitness value and return the result to the master. For the master-slaves scheme to work efficiently, the total communication time t_{c2} between master and slaves must be much smaller than t_{s2}. The T800 Transputer high speed serial link (20 Mbit/s) should be able to cut down t_{c2} substantially. Due to the limited number of T800 transputers available, 3 slaves are used for each NGA. The total number of Transputers used in this PGA implementation is 16.

With one master and 3 slaves, the data on the communication channel will be:

1. the length of individual string times the number of individuals (240 bytes)

2. the input and output measurements of the system (148 bytes)
3. the fitness values of the indiividuals (240 bytes)

The total size of data is approximately 600 bytes.

Assuming direct connection between the master and slaves, (in direct connection, each slave is connected to the master by a dedicated serial link), the communication overhead (task switching) on the Transputer should be small, and the master-slave communication time will then be approximately equal to the transmission time of data on the serial links. With transmission rate of 20 Mbit/s, the master- slaves communication time would be 0.4 ms. Since 0.4 ms is much smaller than the measured GA operators time (10.9 ms), the fitness calculation can be successfully parallelized. Figure E.1 shows the software block diagram of the master-slave scheme.

Due to the limited number of Transputer serial channels, only 1 link on the master node is available for sending and receiving data to and from slave processors. Figure E.2 showed the hardware implementation of the PGA. The communication overhead t_{c2} would increase since direct connection is not possible and data must be re-routed among slave processors. To make sure that in the worst case master-slave communication time would still be small, the timing of processes are measured. The results of the measurement are summarized on Table E.1. It is important to note that the measured t_{c2} is significantly higher than the computed one (1.89 ms vs 0.4 ms). However it is still much smaller than the GA operators time (1.89 ms vs 10.5 ms).

With 1 master and 3 slave processors, the measured speedup factor for level 2 parallelisation is 3.0, which agrees with the calculated speedup factor.

The combined speedup factor for algorithm A and B is:

$$\begin{aligned} Combined\ speedup &= level\ 1\ speedup \times level\ 2\ speedup \\ &= \frac{M}{M_{loc}} \times \frac{\alpha + 1}{\frac{1}{n}\alpha + 1} = 12 \end{aligned} \tag{E.2}$$

The pipeline technique is not used in here because

1. The communication overhead t_{c2} will increase rapidly if direct connection between master-slaves is not possible (pipeline implementation will double the number of slaves)

2. α is relatively small in this example and the difference between pipeline and nonpipeline implementation speedup factor will be small.

GA, PGA with algorithm B, and PGA with combined algorithm A & B achieved speedup factors are listed in Table E.2 for comparison.

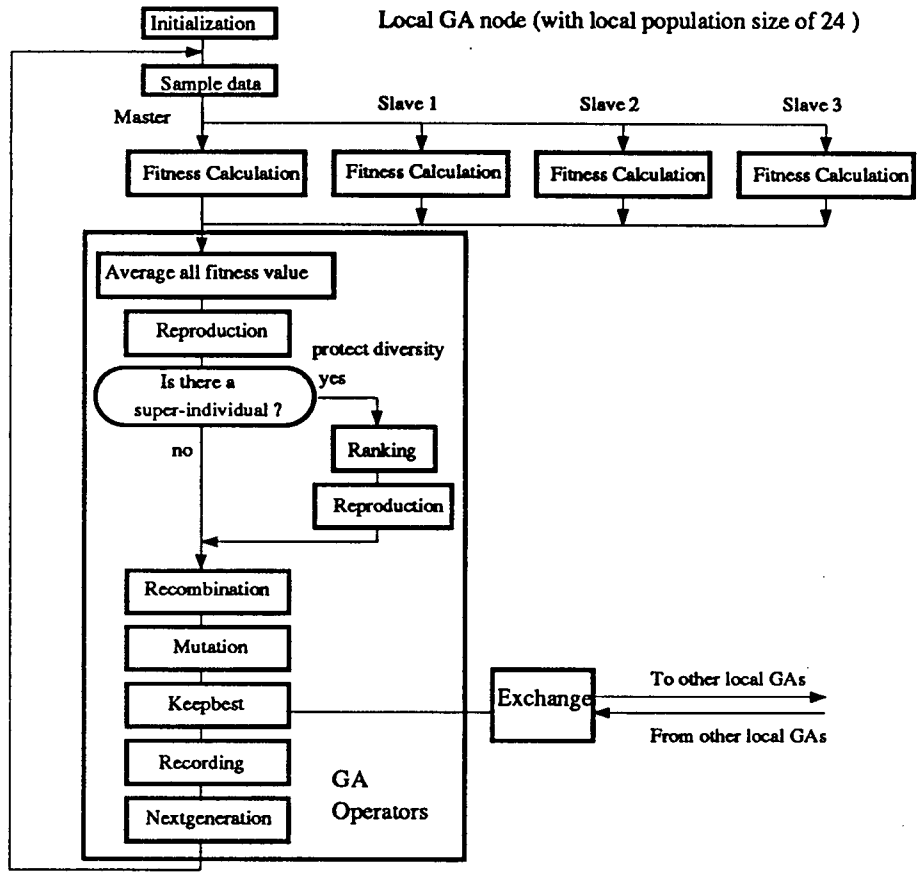


Figure E.1: Software block diagram of the master-slave scheme

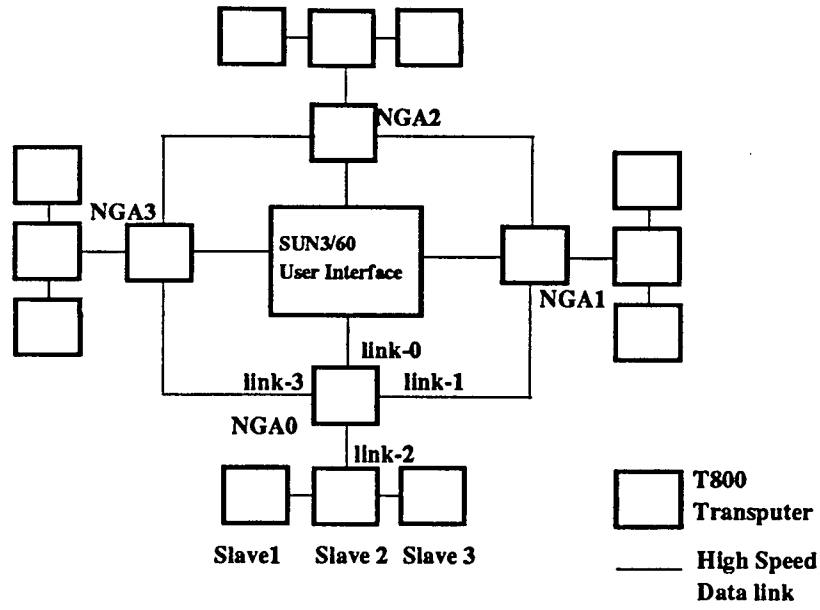


Figure E.2: The Implementation of a PGA (algorithm A and B)

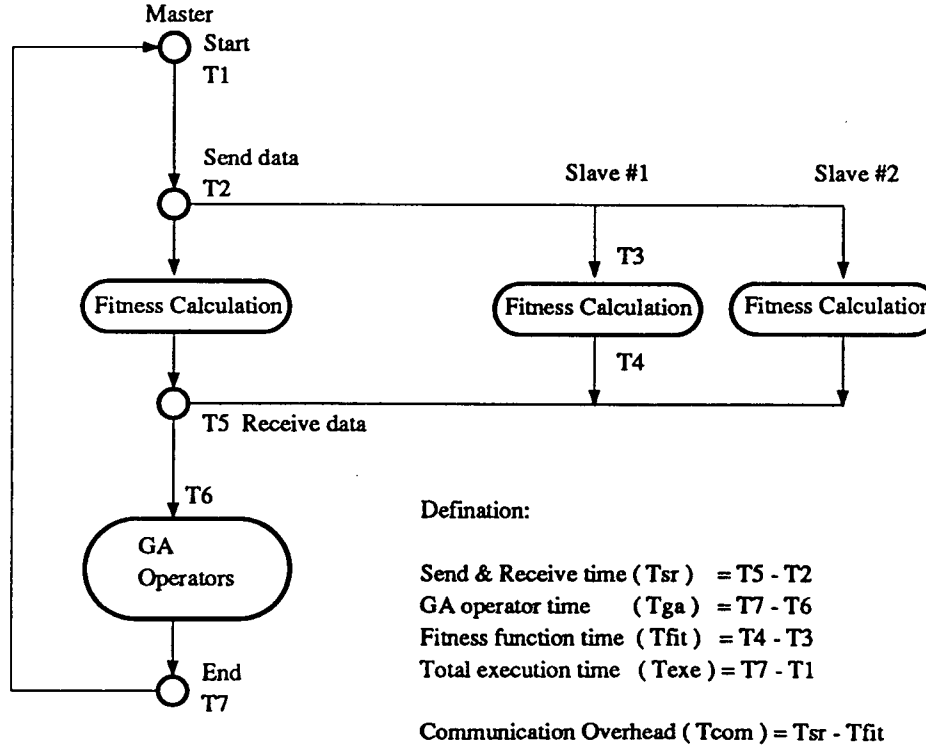


Figure E.3: Measurement Method used for Timing Process

Number of slave processors	0	1	2	3
Execution time (Texe)	97.0 ms	54.0 ms	39.7 ms	32.8 ms
Send - Receive time (Tsr)	86.4 ms	43.3 ms	29.5 ms	22.5 ms
Fitness calculation time (P_{time2})	86.4 ms	43.3 ms	29.0 ms	21.8 ms
Communication Overhead (C_{time2})	0.0 ms	0.6 ms	1.28 ms	1.89 ms
GA operator time (S_{time2})	10.5 ms	10.5 ms	10.5 ms	10.5 ms

Table E.1 Measurement of processes timing

GA and PGA in identifying single link hydraulic compliance :	GA	Level 1 Algorithm (B)	Level 2 Algorithm (A)	Combined Algorithm (A + B)
Number of processors used	1	4	4	16
Global Population size	120	120	N/A	120
Local population size	N/A	30	30	30
Number of generations calculated per sec.	9.8	N/A	N/A	117.6
Speedup achieved	1	4	3	12
Efficiency (Speedup / processors)	1	1	0.74	0.74

Table E.2 The measured speedup factor of various PGAs

References

[1] An and H. Chae. *Model-based control of a robot manipulator*. The MIT Press, Cambridge, Massachusettes, 1988.

[2] B. Armstrong. On finding 'exciting' trajectories for identification experiments involving systems with non-linear dynamics. *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1131–1139, 1987.

[3] An C.H. Atkeson and C.G. Griffiths. Experiment evaluation of feedforward and computed torque control. *Proc. IEEE Int. Conf. Robotics and Automation*, pages 165–168, 1987.

[4] An C.H. Atkeson and Hollerbach. Estimation of inertia parameters of rigid body links of manipulators. *Proc. 24th IEEE Conf. Decision and Control*, pages 990–995, 1987.

[5] R. Das and D. E. Goldberg. Discrete-time parameter estimation with genetic algorithms. *The Modelling and Simulation Conference Proceedings*, 1988.

[6] Edward D. Lazowska Derek L. Eager, John Zahorjan. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–422, March 1988.

[7] David Goldberg. Optimal initial population size for binary-coded genetic algorithms. *TCGA Report No. 85001*, 1985.

[8] David Goldberg. Sizing population for serial and parallel genetic algorithms. *The International Conference on Genetic Algorithms, 89*, pages 70–79, 1989.

[9] David Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[10] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 8:666–677, August 1978.

[11] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[12] K. Kristinsson. Genetic algorithms in system identification and control. Master's thesis, Univ. of British Columbia, Vancouver, B.C., 1988.

[13] K. Kristinsson and G.A. Dumont. Genetic algorithm in system identification. *IEEE International Symposium on Intelligent Control*, 3, 1988.

[14] N. Sepehri G. Dumont P.D. Lawrence and F. Sassani. Cascade control of hydraulically-actuated manipulators. *Robotica*, 8:207–216, 1990.

[15] N. Sepehri P.D. Lawrence and F. Sassani. Gear backlash and stick-slip friction in a teleoperated heavy-duty hydraulic manipulator. *Proc. Canadian Conference on Electrical and Computer Eng.*, pages 45–49, 89.

[16] Crisila B. Pettey Michael R. Leuze and John J. Grefenstette. Genetic algorithms on a hypercube multiprocessor. *Hypercube Multiprocessors*, pages 333–341, 1987.

[17] Crisila B. Pettey Michael R. Leuze and John J. Grefenstette. A parallel genetic algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 155–161, 1987.

[18] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. *The International Conference on Genetic Algorithms, 89*, pages 428–433, 1989.

[19] D. McCloy and H.R. Martin. *The Control of fluid power.* John Wiley and Sons, New York, 1982.

[20] H.E. Merrit. *Hydraulic Control System.* John Wiley and Sons, New York, 1967.

[21] Chrisila C. Pettey and Michael R. Leuze. A theoretical investigation of a parallel genetic algorithm. *The International Conference on Genetic Algorithms, 89*, pages 398–405, 1989.

[22] M.B. Priestley. *Non-Linear and Non-Stationary Time Series Analysis.* Academic Press, New York, 1988.

[23] Nariman Sepehri. *Dynamic Simulation and Control of Teleoperated Heavy-Duty Hydraulic Manipulators.* PhD thesis, Univ. of British Columbia, Vancouver, B.C., 1990.

[24] Reiko Tanese. Parallel genetic algorithm for a hypercube. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms,*

pages 177–183, 1987.

[25] Reiko Tanese. Distributed genetic algorithm. *The International Conference on Genetic Algorithms, 89*, pages 434–439, 1989.