Parallel Genetic Algorithm Fitness Function Team for Eigenstructure Assignment via LQR Designs

João V. da Fonseca Neto
Universidade Federal do Maranhão
Av. dos Portugueses s/n
São Luís - Ma - Brazil 65.080-040
jviana@dmcsi.fee.unicamp.br

Celso P. Bottura
Universidade Estadual de Campinas
LCSI/DMCSI/FEEC - C.Postal 6101
Campinas - SP - Brazil 13.083-970
cpb@turing.unicamp.br

Abstract- The development of a strategy based on a fitness function team exploration for a parallel genetic algorithm and its application to eigenstructure assignment via *LQR* design of a dynamical systems are this works main contributions. A multiobjective optimization based strategy and a decision making framework are formulated in terms of schema theorem and multi-armed bandit problem. An aircraft state space model is used to illustrate the algorithm performance, whose purpose is to find a state feedback controller that leads to a specified eigenstructure assignment.

1 Introduction

The dynamic systems responses are strongly connected to the system's eigenstructures and if the control system designer has suitable and efficient tools that can make an eigenstructure assignment (*EA*) according to design specifications, system's performance can be improved via state feedback laws. A large effort has been spent by researchers to develop methodologies and technics for *EA*, such as: deterministic methods (Sobel and Shapiro, 1985*a*) and (Sobel and Shapiro, 1985*b*) and intelligent exploration of the solution space by genetic algorithms methods, (Liu and Patton, 1996), (Davis and Clark, 1995) and (Clark and Davis, 1997).

To satisfy design specifications, Q and R weighting matrices determination for the linear quadratic regulator (*LQR*) design, that can satisfy the design specifications, is the bottleneck for this form of *EA* problem solution. To overcome this difficulty, a strategy based on a fitness function team and inspired on multiobjective optimization, (Fonseca and Fleming, 1998*a*) and (Fonseca and Fleming, 1998*b*), parallel genetic algorithms, (Koza, 1992), (Holland, 1975) and (Goldeberg, 1989), and decision making framework, (Chankong and Haimes, 1983), is developed to find out these matrices and to benefit from the qualities of *LQR* methodology.

The solution space is searched by several fitness functions, that can be classified as hard and soft. The soft functions are based on inequalities and the hard ones are based on equalities cost functions.

This work is organized as follows. Section (2) presents the multiobjective formulation development. In section (3) the fitness function team concept is established and its features are discussed. Section (4) presents the decisions making framework development, that contains search strategies formulations and together with the genetic algorithm optimizers produces the parallel multiobjective genetic algorithm (*PMOGA*). Section (5) shows genetic algorithm optimizer modeling to target the eigenstructure assignment via *LQR* design. Section (6) comments on parallel implementation characteristics. Section (7) shows *PMOGA* performance obtained from a dynamic system simulation on a workstation network. Finally, the conclusions are presented in section (8).

2 Problem Formulation

Briefly, the eigenstructure assignment via the classic linear quadratic optimization problem is formulated as a multiobjective optimization problem, where the weighting Q and R matrices are independent variables, and further details can be seen at (Bottura and Fonseca Neto, 1999). The aim is to calculate state feedback controller gains that lead to a specified design eingenstructure via the *LQR* problem solution. However, due to the great number of trials that have to be done to find the Q and R matrices, which must satisfy design specifications and maintain the *LQR* asymptotic guaranteed stability margins, genetic algorithms are utilized to search these matrices. Given a linear time invariant system:

$$\dot{x} = Ax + Bu \tag{1}$$
$$y = Cx$$

the control vector is given by :

$$u = -K(Q,R)x \tag{2}$$

where K is the gain matrix $(m \times n)$, Q is the positive semi-definite state weighting matrix, R is the positive semi-definite control weighting matrix.

The controller gains $K(Q,R)$ are given by the Algebraic Riccati Equation (*ARE*) solution's for the LQR problem, where the control law u is found, equation (2), when the minimization of the quadratic performance cost, equation (3), subject to restriction (1) is performed.

$$J = \int_0^\infty [x^T Q x + u^T R u] dt \tag{3}$$

0-7803-5536-9/99/$10.00 ©1999 IEEE

The eigenstructure assignment problem, from this point of view, consists on the gain matrix $K(Q,R)$ determination that imposes the desired closed-loop system:

$$\dot{x} = (A - BK(Q,R))x \quad (4)$$

The closed-loop system spectrum range of (4) must satisfy the design specifications and the left and right eigenvectors must satisfy eigenvalue sensitivity restrictions. The multiobjective optimization problem (MOP) formulation, that allows the determination of a controller $K(Q,R)$ through application of genetic algorithms search technics to solve the eigenstructure problem, is obtained by joining the *LQR* problem solution and the eigenstructure restrictions (closed-loop system and eigenvalue sensitivity spectrum bounds). The *MOP* formulation in a normalized form is:

$$\min_{Q,R} \sum_{i=1}^{n} s_i(Q,R) \quad (5)$$

s.t.

$$s_i(Q,R) \leq 1 \quad i = 1,...,n \quad (6)$$

$$\lambda_{ei} \leq \lambda_{ci}(Q,R) \leq \lambda_{di} \quad i = 1,...,n \quad (7)$$

where $s_i(Q,R) = (\frac{\|L_i(Q,R)\|_2 \|R_i(Q,R)\|_2}{< L_i(Q,R) R_i(Q,R) >})/\epsilon_i$ is the normalized eigenvalues sensitivity and the i-th design specification $\epsilon_i > 0$; $\|L_i(Q,R)\|_2$ e $\|R_i(Q,R)\|_2$ are the 2-norm of the left and right eigenvectors, respectively, and $< L_i(Q,R)R_i(Q,R) >$ is the eigenvectors dot product. λ_{ei} and λ_{di} are the left and the right i-ths eigenvalues bounds, respectively, for the i-th desired eigenvalue λ_{ci}.

3 Fitness Function Team

The fitness function team is defined as a fitness function set, $FF_{team} = (ff_1, ff_2, ..., ff_n)$, that is used to select genetic algorithms (*GA*) individuals that will assemble permanent populations generations. The cost function, to choose permanent population, and a previous selection criterion, to choose suitable candidates from a transient population that comes from one *GA* search iteration, define each ff_j, $j = 1,...,n$, structure.

Functions of the set can compete between each other to explore the whole search space or can be allocated to explore selected regions; these search space regions can be established by niche induction methods. The individuals selections forms are the main differences between the FF_{team} elements. The restrictions quantity, type and the cost function nature make some of these elements to be considered harder than others.

Five types of fitness functions structures were developed in this research. However, only two cost functions constitute the FF_{team} core; one of them has its origin in the inequalities method that is the eigenvalues sensitivity cost function (5), called J_S, and the other is cost function (8) used by (Davis and Clark, 1995) and developed by (Liebst and Huckabone, 1992), called eigenstructure assignment equality and called J_E.

$$\sum_{i=1}^{n} f_{\lambda i}(\lambda_{ei}-\lambda_{ci})^*(\lambda_{ei}-\lambda_{ci})+(\vec{v}_{ei}-\vec{v}_{ci})^* F_{vi}(\vec{v}_{ei}-\vec{v}_{ci}) \quad (8)$$

where λ_{ei} i-th specified eigenvalue, λ_{ci} i-th desired eigenvalue, f_i i-th eigenvalue weighting, $\vec{v}_{ei}$ i-th specified eigenvector, $\vec{v}_{ci}$ i-th desired eigenvector, F_{vi} i-th eigenvector weighting diagonal matrix.

The cost functions and the previous selection criterion, that are the eigenvalues sensitivity restrictions (s_iR), relation (6), and eigenvalues range restrictions (λ_{ci}R), relation (7), combination furnish fitness functions structures. For instance: ff_1 is formed by cost function, equation (5), the eigenvalues sensitivity restrictions that are used based on Pareto's optimality criterion (POC) and eigenvalues range restrictions. The ff_2 uses the same cost function and restrictions, but it doesn't follows POC . An individual's incorporation in new populations, when previous selections based on Pareto's optimality criterion is not used, will be accepted if each individual's s_i is smaller than the greatest individual's s_i of the current population. This criterion is referenced as individual greater s_i criterion (IGs_iC). Table (1) shows the FF_{team} structure developed and implemented in this work.

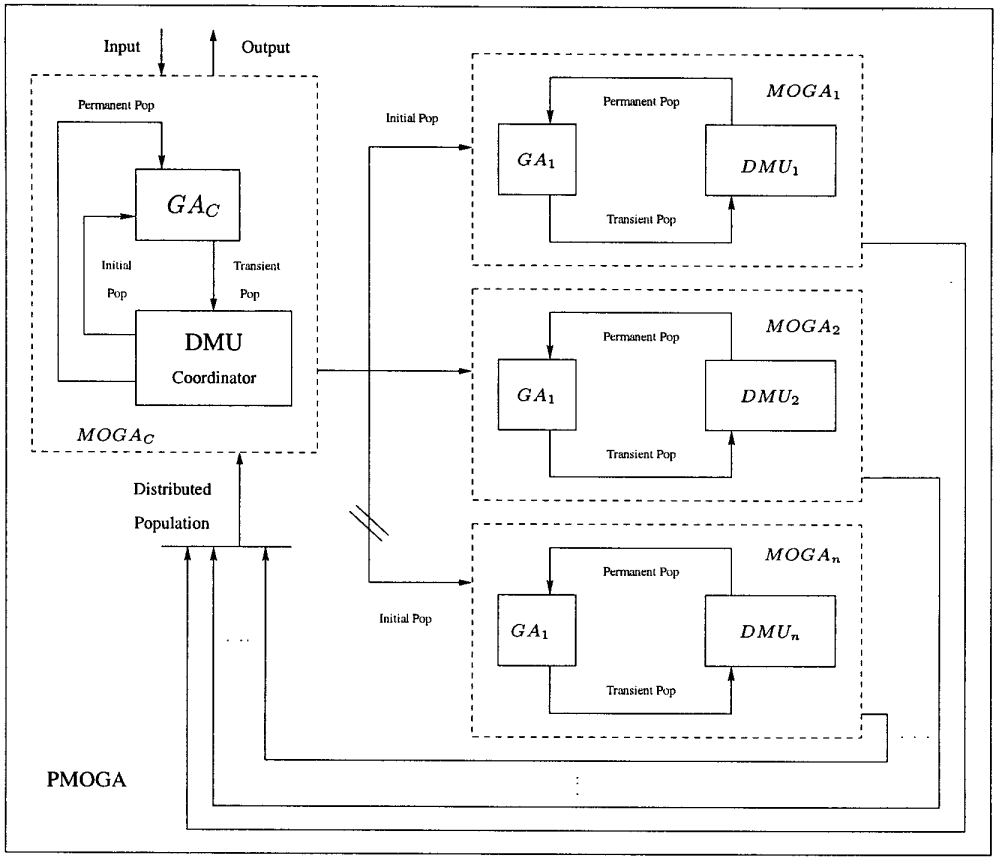
	ff_1	ff_2	ff_3	ff_4	ff_5
Cost function	J_S	J_S	J_E	J_E	J_E
Previous Selection Criterion	s_iR	s_iR	s_iR	s_iR	
	λ_{ci}R	λ_{ci}R	λ_{ci}R	λ_{ci}R	λ_{ci}R
	POC	IGs_iC	POC	IGs_iC	

Table 1: FF_{team} basic structure

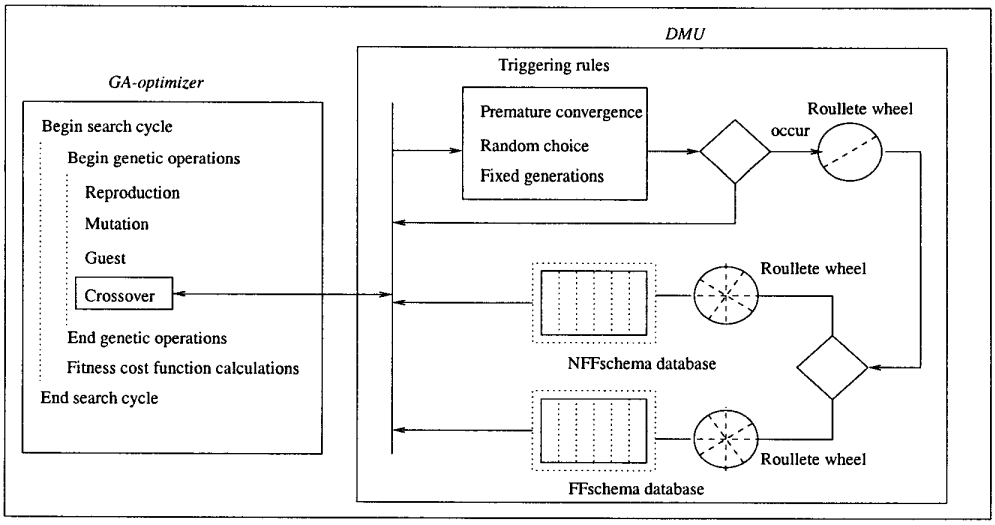
4 Decision Making Unity

The decision making unity (*DMU*) is a logical framework, where strategies formulations occur, that will guide the genetic algorithm optimizer in the direction of an intelligent search. The chosen decisions are based on fitness cost functions (5) and (8), Pareto's optimality criterion, schema theory and multi-armed bandit paradigm. The following paragraphs describe *DMU* strategies, considering a parallel genetic algorithm on a distributed environment.

The logical framework is explained in three levels. First in a high level, using Figure (1a) that represents *DMU*'s and *MOGA*'s interactions on the distributed environment. The second level, using Figure (1b), that represents *DMU* action strategies and the point of interaction with the *GA*-optimizer. The third level, using Figure (2), is the lowest one and shows how the databases are accessed, how the Q or R feasible fitness schema (*FFschema*) or non-feasible fitness schema (*NFFschema*) type is defined and how crossover operation is performed when the strategies are triggered.



a) Parallel *MOGA* - *DMU* and *GA*-optimizer interactions.



b) *MOGA* - *DMU* strategies and *GA*-optimizer crossover operation interactions.

Figure 1: PMOGA interactions on a distributed environment

The *DMU's* and GA's optimizers are the multitobjective genetic algorithm (*MOGA*) basic elements and each basic element pair constitute a *MOGA*. The *MOGA* set is called parallel *MOGA*, $PMOGA = (MOGA_c, MOGA_1, MOGA_2, ..., MOGA_n)$, in a distributed environment, where $MOGA_c$ is the coordinator and the $MOGA_i, i = 1, n$, are the coordinateds.

Firstly, the *DMU's* are fed by an initial population (randomly generated), Figure (1a); this population after an evaluation by the *DMU* coordinator (DMU_C) is sent immediately to the distributed GA-optimizers, $OPTI_D = (GA_1, GA_2, ..., GA_n)$, where each GA_i makes one search and generates a transient population to feed their own DMU_i that belongs to the distributed DMU_D set, $DMU_D = (DMU_1, DMU_2, ..., DMU_n)$. Each DMU_i takes decisions to guide the search, assembles a new population and sends it to his GA_i-optimizer. These processes go on until any stopping criterion is reached.

Each *DMU* has his own decision strategy. For instance, DMU_1 has individuals (solution points) selection criterion to assemble populations based on POC and fitness cost function (5) is used to check its performance. The DMU_2 doesn't have individual selection based on POC and its efficiency is verified with the fitness function used by DMU_1.

If one of *the DMU's*, that belongs to DMU_D set, detects that any of the stopping criteria (convergence or maximum iterations) is satisfied, the current population is sent to the DMU_C. The distributed *MOGA's* populations are sent to the DMU_C in an asynchronous communication mode. After receiving all distributed populations and having finished his own search, the DMU_C assembles a new population with distributed environment best individuals and evolves this population.

The schema theorem (*STheorem*) and the multi-armed bandit paradigm (*MParadigm*) are incorporated in all *DMU's*. The schema theorem can be used to avoid schemata proliferation and to increase population variety. The *MParadigm* is used to extract strength potentialities that can exist on the weakest ones. Two basic elements build the *DMU* framework. One of them is a selection criteria that defines the manner and how many individuals are inserted into the permanent population. The second element manages the schema theorem and multi-armed strategies; these strategies actions are based on three triggering rules: the first one acts on population or best individual premature convergence detection, the second one acts based on small probability of occurrence, the third one acts in a deterministic manner and periodically between generations life and death.

The DMU *STheorem* and *MParadigm* strategies, Figure (1b), are based on random choices, implemented as *roullete wheel*, (Goldeberg, 1989), of individuals and decisions, triggering rules and two databases of QR individuals that are assembled during the search cycle. One database, *FFschema* database, is assembled with the best individuals, i.e., the ones that have presented the best fitness function values during the search. The second database, *NFFschema*, is assembled with the worst individuals. Both databases are assembled during the search cycle.

These strategies act together with crossover (*X-over*) operation, Figure (1b), in the sense that *X-over* executes the defined strategy. Every time, before the *X-over* operation is performed, *DMU* checks three events occurrence; these events are referred as triggering rules. If one of the events has occurred a random choice is made to define the individual type, *FFschema* or *NFFschema*, that will be combined with a random chosen individual of the permanent population. After the individual type definition, the next step is a randomly picking up of one individual on the respective database. The last step is to send the chosen individual to be combined in the *GA*-optimizer.

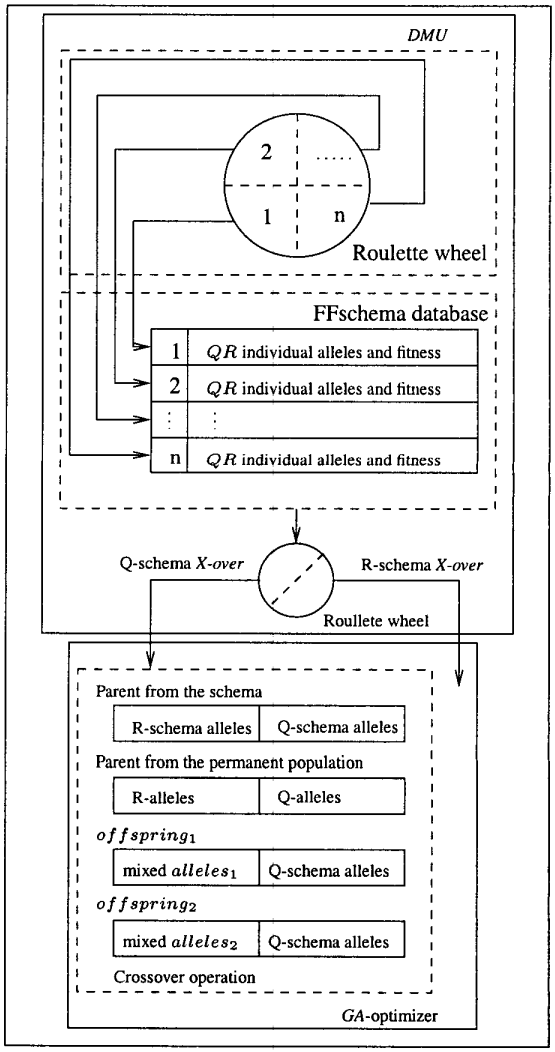


Figure 2: The schema individual choice and crossover operation

After the random definition of *FFschema* or *NFFschema* strategy, Figure (1b), two random choices are made, Figure (2). Supposing that the defined strategy was *FFschema*, one QR individual is picked up from the *FFschema* database randomly, which one is chosen with a biased roulette wheel. After that, the roulette wheel is used, once again, to define a Q or a R schema type for the QR-individual; in this case it was a Q type. As a result of this two choices, a QR-individual schema is presented to the *GA*-optimizer and the schema type is Q-alleles. The chosen individual is sent to the *GA*-optimizer and the results of the crossover schema operation are two *offsprings* with the same Q-alleles and two different mixed R-alleles that are obtained combining schema individual's R-alleles and permanent population individual's R-alleles.

5 Genetic optimizer

The matrices Q and R genetic modeling, genetic operations and fitness function team (calculations and ordering) are the GA optimizer basic elements presented in this section. The matrices modeling and genetic operations are all numerically performed on a decimal basis.

5.1 Matrices Q and R Genetic Modeling

The Q and R matrices model developed in (Bottura and Fonseca Neto, 1999) is used, with the change that chromosomes alleles are not represented by using 0's (zeros) and 1's (one's) strings, but they are represented by decimal numbers. Each matrix Q and R is represented by one chromosome and matrices Q and R pairs are called QR-individuals and a set of QR pairs comprises a population. No genetic operations between Q and R alleles are allowed, as they belong to different species.

5.2 Genetic Operations

Two sequential steps constitutes the genetic operations, that are: chromosome operations and evaluations of individuals generated from those operations. The first step, chromosomic operations are reproduction, crossover, mutation and guest. The second step, fitness function calculation and ordering provide informations to the decision unity; subsection (5.3) presents these operations algorithms.

The reproduction operation doesn't need to be evaluated because this operation just verifies those individuals that have the worst fitness which are removed and replaced by a clone of the population strongest individuals. The reproduction occurrence probability is 10%.

The algorithm for crossover operations is single-point (*SPX*) crossover; chromosome combinations are made between alleles only and combination degree is adjusted according to population age. The crossover operations happen with probability of 60%.

Two decimal mutation algorithms were designed, one of them is global and the other is local; they were inspired on (Marrison and Stengel, 1997). The local mutation is an individual multiplicative change and the global mutation is an individual incremental change; this increment is calculated from the fitness function value for the strongest and the weakest population's individuals. The mutation occurrence probability amongst all operations is 15% and between global and local is 50% for each one.

The guest operator inserts new individuals that do not have any relationship with the current population. This operator purpose is to untrack some populations of saturation levels and to allow a better exploration of the search space. The guest occurrence probability is 5%.

5.3 Fitness function structure

In this approach the fitness function provides information to DMU's. The fitness function has the aim to perform cost functions calculations and to sort individuals based on some criteria.

Before the population's individuals evaluation starts, some parameters of these functions are initialized with the initial population's individuals and these parameters are determined before this population's distribution. The initialized parameters are: normalized performance function vector, $\Delta_{current} = (s_1, s_1, ..., s_n)_{current}$, normalized performance function sum, $E_{current}$, eigenstructure assignment equality, $J_{current}$, and controller, $K_{current}$, to each individual.

The fitness functions select candidates for future populations; they are tagged as current values and are stored in a transient population vector. The new individuals quantity that will be part of the permanent population are randomly chosen. The basic rule for individuals insertion in the permanent population is: if some individuals of the transient generation are better than any best current individuals, according to the DMU strategy, they will assume current positions. Thus, the fitness functions algorithm k-th iterative step is:

1. Normalized performance functions vector calculation:
   - For ff_j , $j = 1, ..., 4$

     $$\Delta^k = (s_1(K^k), s_2(K^k), ..., s_n(K^k)) \qquad (9)$$

   where $s_i(K^k) = \frac{S_i(K^k)}{\epsilon_i}$, $i = 1, 2, ..., n$
2. Cost functions calculations:
   (a) Normalized performance cost function sum:
   - For ff_j , $j = 1$ and 2

     $$E^k = \sum_{i=1}^{n} s_i(K^k), \quad i = 1, 2, ..., n \qquad (10)$$

   (b) Eigenstructure assignment equality cost function:
   - For ff_j , $j = 3, ..., 5$

     $$J^k = J(K^k) \qquad (11)$$

3. Restrictions bounds ordering:

- All ordering criteria consider eigenvalues limits:

$$\lambda_{ei} \leq \lambda_{ci}(K^k) \leq \lambda_{di}, \quad i = 1, ..., n \qquad (12)$$

(a) Maximum $s_i(K^k)$ ordering:

- For ff_j , $j = 1$ and 3

$$(a1) \quad \max_{K^k}\{s_i(K^k)\} < \max\{s_i\}_{current}, \quad i = 1, 2, ..., n \qquad (13)$$

or

- For ff_j , $j = 1$

$$(a2) \quad \max_{K^k}\{\Delta^k\} = \max\{\Delta_{current}\} \quad and \quad E^k < E_{current} \qquad (14)$$

- For ff_j , $j = 3$

$$(a3) \quad \max\{\Delta^k\} = \max\{\Delta_{current}\} \quad and \quad J^k < J_{current} \qquad (15)$$

(b) $s_i(K^k)$ Pareto's optimality ordering:

- For ff_j , $j = 2$ and 4

$$(b1)\ s_i(K^k)\} \leq \Delta^i_{current}, \quad i = 1, 2, ..., n \qquad (16)$$

or

- For ff_j , $j = 2$

$$(b2)\ \Delta^k = \Delta^i_{current} \quad and \quad E^k < E_{current}, \quad i = 1, 2, ..., n \qquad (17)$$

- For ff_j , $j = 4$

$$(b3)\ \Delta^k = \Delta^i_{current} \quad and \quad J^k < J_{current}, \quad i = 1, 2, ..., n \qquad (18)$$

(c) Eigenvalues bounds ordering:

- For ff_j , $j = 5$

$$J^k < J_{current} \qquad (19)$$

6 MOGA distributed parallelization

The parallel multiobjective genetic algorithm (*PMOGA*) was developed and implemented based on an integration between decision making units, section (4), and GA-optimizers, section (5), dedicated to the eigenstructure assignment via linear quadratic regulator problem. The parallel genetic model used is based on independent runs parallelization, (Koza, 1992), where each network processor evolves the same initial population and the programming paradigm used is based on *SPMD* (*single-procedure multiple-data*), where each network processor runs the same program. The characteristic of running the same program on several network processors makes the *SPMD* paradigm well suitable for the *MOGA-LQR* implementation, because each processor solves the same problem, but each one gives a different search solution treatment for the same initial population. This paradigm macroelements are: the coordinator, that makes communication among processors and one *LQR* search, and several coordinates, that make only *LQR* problems serial processing. There are only two communication forms; the first, communication from only one processor to all processors, before the *LQR* problem solution, and the second, communication from all processors to only one processor, after the *LQR* problems solution.

The algorithm termination happens when restrictions or cost functions can not be improved any more or the specified eigenstructure can not be reached or the random search process reaches its iterations maximum limit.

7 Results

A state space variable model was used to study the *PMOGA* performance. The system represents a Lockheed aircraft, L1011 Tristar type. The A, B and C matrices of equation (1) set and the design specifications are given in (Bottura and Fonseca Neto, 1999), and they were obtained from (Davis and Clark, 1995) and (Sobel and Shapiro, 1985*b*).

Two kinds of results are analyzed. The first concerns *DMUs* strategies and the *GA*-optimizers search performances; these results are presented using graphics of populations and individuals profiles for fitness evolution and populations profile. The second result type shows controllers performances when implemented on the state space model, equation(1), and a impulse signal is applied to its input.

7.1 Simulations

A Sparc-SUN network was the computation environment used for simulations. The results presented came from five tasks; each task has his own fitness function, where one of the tasks (coordinator) creates the initial population and distribute it among the other tasks (coordinateds). The permanent population is built up of 10 individuals (controllers), i.e., for each new generation only 10 individuals survive to the conditions imposed by the fitness functions. The transient population size depends on genetic operation type that is random; the crossover operation inserts 14 individuals into the population, the mutation inserts 5, reproduction inserts 2 and guest 2; hence, the transient population might have 24 or 15 or 12 individuals.

Figures (3-5) present the algorithm behaviour for 100 parallel genetic search trials to find feasible QR-individuals. Figures (3) and (4) are analyzed together, for fitness function ff_1 every time that there is a reduction on $s_i(K)$ there is too a reduction on the cost function, because its nature is Pareto's dependent; however this phenomena does not happen

with the other four. It can be seen that all five search found out a feasible eigenvector solution point. Figure (5) shows final population profile and the best individuals are chosen to assemble a new population, where a search refinement based on Pareto is made.

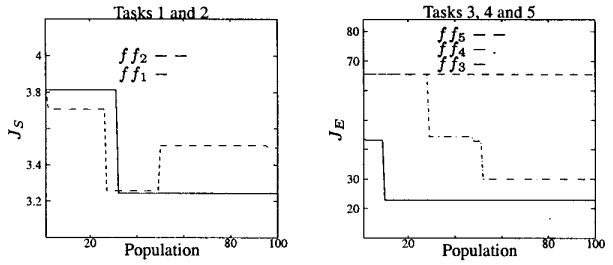


Figure 3: Populations × $FFteam$ cost function behaviour for the best individual.

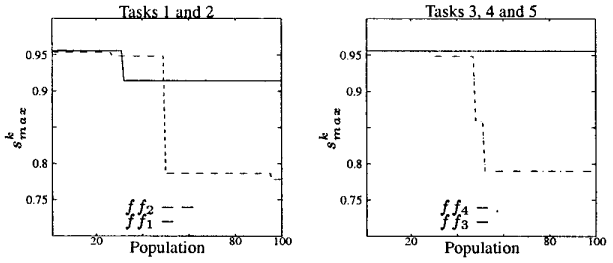


Figure 4: Populations × ff_1, ff_2, ff_3, ff_4 maximum s^k.

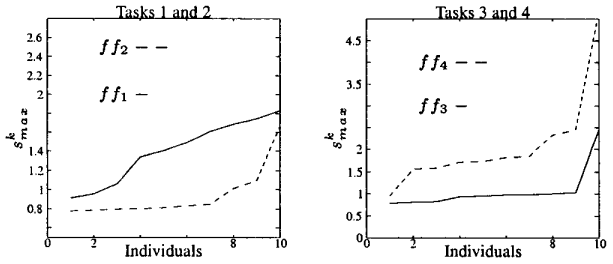


Figure 5: Final population individuals profile × ff_1, ff_2, ff_3 and ff_4 maximum s^k.

Figure (6-8) shows the designed controllers performances compared with the basic controller. As can be seen the controllers designed by the proposed methodology present good performance to impulse signals.

Figures (9) and (10) present the sequential refinement search evolution results. It can be seen that the best individual and the population profile were improved, Figure (9). The sequential controller also presents a good performance in terms of the impulse response, Figure (10).

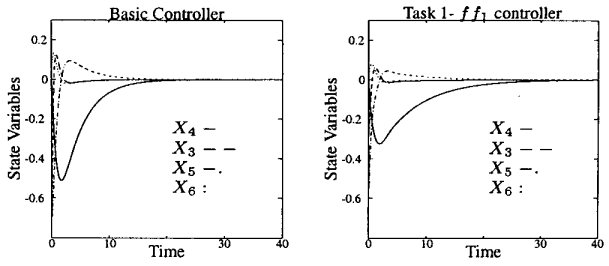


Figure 6: Impulse response for the basic controller and ff_1 controller.

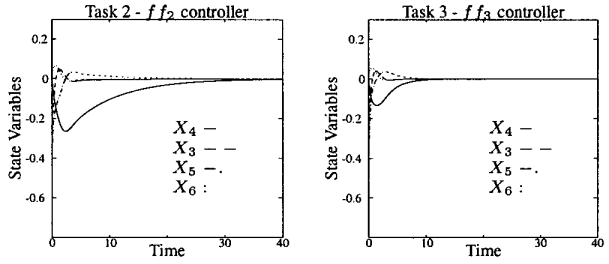


Figure 7: Impulse response for ff_2 controller and ff_3 controller.

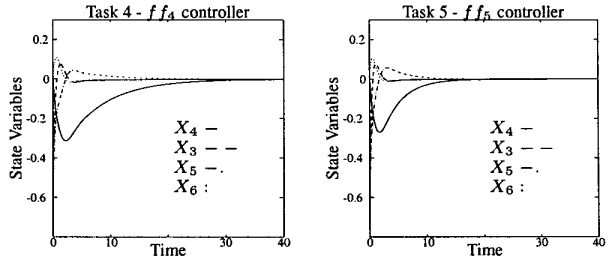


Figure 8: Impulse response for ff_4 controller and ff_5 controller.

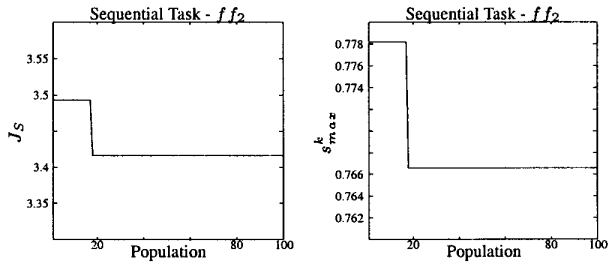


Figure 9: Sequential task - population × ff_2 cost function and maximum s^k.

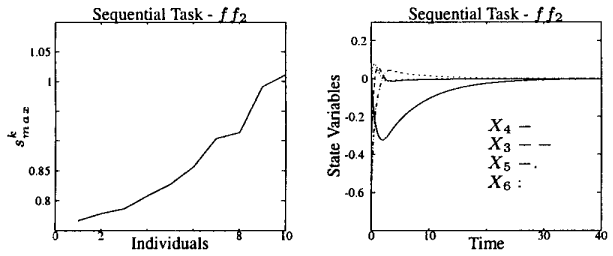


Figure 10: Sequential task - final population individuals profile $\times$ ff_2 maximum s^k and impulse response for ff_2 controller.

8 Conclusion

The parallel genetic algorithm together with a fitness functions team, *DMU* based on schema theorem and multi-armed bandit paradigm, small population and guest operator has shown to be an efficient tool for controller design when the entire system eigenstructure performance has to be assigned.

The *PMOGA* found a good quality solution and a small amount of time was spent to find the required *EA*. However, a much better performance would be obtained if the *DMU* had knowledge of the relevant system's behaviour for variables that must be controlled, because to get a feasible solution the designer has to make manual adjustments on QR-individual numerical size and on the multiplicative crossover operations parameter that varies with population's age. So an intelligent *DMU* should be the next step to get *PMOGA* improvements.

The coarse grained communication among processors and the genetic parallel model based on independent runs made the *SPMD* programming paradigm very suitable for solving the *EA* problem via *LQR* design, because the computing solution is formulated based on parallel and independent *LQR* design solutions. Besides that, the distributed solution can be easily expandable for a greater number of different kinds of fitness functions structures among network processors.

Bibliography

Bottura, C. P. and J.V. Fonseca Neto (1999). Parallel eigenstructure assignment via lqr design and genetic algorithms. *1999 American Control Conference, accepted paper.*

Chankong, Vira and Yacov Y. Haimes (1983). *Multiobjective Decsion Making: Theory and Methdology*. Vol. 8 of *8*. Elsevier Science Publishing Co, Inc. P.O. Box 211, 1000 AE Amsterdam, Netherlands.

Clark, T. and R. Davis (1997). Robust eigenstructure assignment using the genetic algorithm and constrained state feedback. *IMechE* **211**(Part-I), 53–61.

Davis, R. and T. Clark (1995). Parallel implementation of a genetic alghorithm. *Control Eng. Practice* **3**(1), 11–19.

Fonseca, C.M. and P.J. Fleming (1998*a*). Multiobjective optimization and multiple constraint handling with evolutionary algorithms-part i: A unified formulation. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans* **28**(1), 26–37.

Fonseca, C.M. and P.J. Fleming (1998*b*). Multiobjective optimization and multiple constraint handling with evolutionary algorithms-part ii: Application example. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans* **28**(1), 38–48.

Goldeberg, David E. (1989). *Genetic Algorithms in Search, Optimization ,and Machine Learning*. Addison Wesley Publishing Company. Ann Arbor-Michigan-USA.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press . Ann Arbor-Michigan-USA.

Koza, John R. (1992). *Genetic Programmming : On the Programming of Computers by Means of Natural Selection*. The MIT Press. Cambridge, Massachusetts - USA.

Liebst, B.S. and T.C. Huckabone (1992). An algorithm for robust eigenstructure assignment using linear quadratic regulator. *AIAA Guidance Navigation and Control conf* pp. 896–909.

Liu, G.P. and R.J. Patton (1996). Robust control design using eigenstructure assignment and multiobjective optimization. *International Journal of Systems Science* **27**(9), 871–879.

Marrison, C.I and R.F Stengel (1997). Robust Control System Design Using Random Search and Genetic Algorithms. *IEEE Tranaction on Automatic Control* **42**(6), 835–839.

Sobel, K. M. and E. Y. Shapiro (1985*a*). Eigenstructure assignment : A tutorial part I Theory. In: *Proceedings of American Control Conference* 5^{th}. Vol. 1. Saint-Nazaire, USA. pp. 456–460.

Sobel, K. M. and E. Y. Shapiro (1985*b*). Eigenstructure assignment : A tutorial part II Applications. In: *Proceedings of American Control Conference* 5^{th}. Vol. 1. Saint-Nazaire, USA. pp. 461–462.