# Planificación Difusa auto-adaptativa para hebras

Raúl Tremsal, Pedro Furlani y Carlos Kavka
E-mail: ckavka@unsl.edu.ar
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - 5700 - San Luis

**Abstract**

El modelo de programación con hebras es muy utilizado hoy en día. Las hebras se pueden ejecutar en forma concurrente dentro del mismo programa, ejecutando tareas claramente divididas, y compartiendo datos. Las bibliotecas estándar de hebras proveen control sobre la planificación, pero no le permiten al usuario definir nuevos algoritmos de planificación.

En este trabajo se propone dar al programador de aplicaciones que utilizan hebras la posibilidad de definir nuevos algoritmos de planificación utilizando reglas difusas. Se propone además la utilización de un algoritmo de auto-adaptación para mejorar la calidad del algoritmo en forma automática.

## 1. Introducción

A través de la utilización de hebras se puede dividir un programa en subtareas cuya ejecución puede ser llevada a cabo en forma concurrente. Un algoritmo de planificación selecciona una hebra para su ejecución a partir de un conjunto de hebras que están esperando su turno. Esta decisión está determinada por un algoritmo de planificación y se basa en la prioridad de las hebras. El programador puede definir las prioridades y seleccionar una política particular, pero no puede definir un nuevo algoritmo de planificación.

En este trabajo, se propone mejorar la biblioteca estándar de hebras con la inclusión de un mecanismo para la definición de nuevos algoritmos de planificación en base a reglas difusas. Los algoritmos obtenidos son mucho más fáciles de entender y son más robustos.

Un problema que surge, sin embargo, es que el proceso de 'sintonía fina' o adecuación, a veces, se transforma en un proceso de corrección por intento y error. Se propone la utilización de una técnica de aprendizaje por refuerzo combinada con redes neuronales para mejorar el desempeño del planificador.

En este trabajo se muestra como ejemplo, la definición de un planificador de tiempo real para hebras, y los resultados al aplicar el proceso de mejora.

## 2. Planificación difusa

Los algoritmos de planificación para hebras involucran usualmente dos atributos por cada hebra:

- prioridad: determina que hebra puede acceder al procesador.
- política: es una forma de especificar de que forma dos hebras de la misma prioridad se ejecutan y comparten la CPU.

Las dos políticas principales son

- FIFO: permite a las hebras que se ejecuten hasta que terminen o se bloqueen. Cuando una hebra se bloquea, es colocada al final de su cola de prioridades.
- Round-Robin: permite que las hebras se ejecuten sólo por una cantidad limitada de tiempo. Cuando su tiempo se acaba, el procesador se asigna a otra hebra.

~~Se agregó a la biblioteca estándar de hebras un nuevo algoritmo denominado FUZZY:~~ Trabaja como el Round-Robin, pero las prioridades de las hebras se definen por reglas difusas, basadas en atributos definidos por el usuario. Los atributos pueden estar basados en el tiempo, como el tiempo de comienzo o fin (deadline) esperado, o ser independientes del tiempo, como por ejemplo la importancia de una hebra, su costo, etc.

De esta forma, la definición de un algoritmo de planificación involucra:

- La definición de un conjunto de atributos y sus correspondientes valores lingüísticos.
- La definición de un conjunto de reglas utilizadas para calcular la prioridad de una hebra teniendo en cuenta sus valores de atributos.

Por ejemplo, se puede definir un planificador de tiempo real considerando tres atributos:

- tiempo de comienzo: las hebras se planifican únicamente sólo después de su tiempo de inicio.
- criticalidad: las hebras se planifican sólo si no hay hebras más importantes esperando para ejecución.
- tiempo de finalización: las hebras deben finalizar antes de este tiempo.

Se asume que los tiempos de llegada y de finalización de las hebras no se conocen hasta que el programa es puesto en ejecución. De esta forma, no es posible utilizar un algoritmo de planificación estático. El algoritmo propuesto es dinámico y la planificación se realiza en línea. Cada vez que llega una hebra o cada vez que alguna finaliza su ejecución, las prioridades se recalculan.

Los valores lingüísticos para cada atributo son:

- tiempo de comienzo: temprano, medio y tardío.
- criticalidad: importante, promedio y no-importante.
- tiempo de finalización: temprano, medio y tardío.

Los valores de los atributos para cada hebra son transformados en valores difusos, y luego se utiliza un conjunto de reglas para calcular los valores de prioridad. Un conjunto de reglas difusas para este planificador de tiempo real se muestra a continuación:

| inicio | finalización | criticalidad | prioridad |
|---|---|---|---|
| - | temprano | importante | diez |
| temprano | temprano | promedio | nueve |
| medio | temprano | promedio | ocho |
| tardío | temprano | no importante | siete |
| temprano | temprano | no importante | ocho |
| medio | temprano | no importante | siete |
| tardío | temprano | no importante | seis |
| temprano | medio | importante | seis |
| medio | medio | importante | cinco |
| tardío | medio | importante | cuatro |
| - | medio | promedio | cinco |
| - | medio | no importante | cuatro |
| - | tardío | importante | tres |
| - | tardío | promedio | dos |
| - | tardío | no importante | uno |

Por ejemplo, la segunda fila de la tabla representa la regla difusa:

| | |
|---|---|
| Si | el tiempo de comienzo es temprano y<br>el tiempo de finalización es temprano y<br>la criticalidad es promedio |
| Entonces | la prioridad es nueve. |

Los valores del centro corresponden al centro de las funciones triangulares, y los otros dos valores a la amplitud izquierda y derecha respectivamente. La palabra abierto quiere decir que en esa dirección, el valor de pertenencia es 1 desde la posición central.

De esta forma, cada hebra recibe un valor de prioridad calculado en base a los atributos. Las hebras se planifican en base a estas prioridades calculadas en forma dinámica.

Los valores de prioridad también se definen como conjuntos difusos. Algunos valores para poder definir ejemplos se muestran a continuación:

| variable | conjunto difuso | centro | amplitud izquierda | amplitud derecha |
|---|---|---|---|---|
| | temprano | 100 | abierto | 150 |
| tiempo de inicio | medio | 300 | 150 | 150 |
| | tardío | 500 | 150 | abierto |
| | temprano | 100 | abierto | 150 |
| tiempo de finalización | medio | 300 | 150 | 150 |
| | tardío | 500 | 150 | 150 |
| | no importante | 2 | abierto | 3 |
| criticalidad | promedio | 5 | 3 | 3 |
| | importante | 10 | 3 | abierto |
| | uno | 1 | abierto | 1 |
| | dos | 2 | 1 | 1 |
| | tres | 3 | 1 | 1 |
| | cuatro | 4 | 1 | 1 |
| prioridad | cinco | 5 | 1 | 1 |
| | seis | 6 | 1 | 1 |
| | siete | 7 | 1 | 1 |
| | ocho | 8 | 1 | 1 |
| | nueve | 9 | 1 | abierto |

Este algoritmo de planificación de tiempo real es sólo un ejemplo. Un algoritmo completamente diferente se puede definir involucrando atributos tales como costo, beneficio, etc. Las reglas correspondientes se usarán para calcular los valores de prioridades, y las hebras entonces se planifican de acuerdo a esos valores obtenidos.

## 3. Auto adaptación

El objetivo de esta propuesta es mejorar la calidad del planificador difuso. El planificador se evalúa bajo condiciones controladas. Sólo se hacen cambios cuando su desempeño no es adecuado.

Condiciones controladas quiere decir casos de entrenamiento, o en otras palabras, un conjunto de hebras que incluye casos límite orientados a generar dificultades en la planificación. Para el ejemplo de tiempo real, un caso de entrenamiento estará constituido por hebras que permitan planificaciones exitosas y planificaciones no exitosas también.

Para la etapa de entrenamiento, proponemos representar al planificador difuso como una red neuronal. Se ha seleccionado la componente ASN (Action Selection Network) del modelo GARIC (Generalized Approximate Reasoning-based Intelligent Control) propuesto por Berenji y Khedkar. Si se conoce la salida esperada del sistema, es decir, la prioridad correcta para cada hebra, es muy fácil corregir los conjuntos difusos representados en la red.

Este es, sin embargo, el principal problema, dado que este valor no se puede conocer. Se propone utilizar un tutor para la etapa de entrenamiento, basado en una clase de estrategia

de aprendizaje por refuerzo denominado Q-learning. El algoritmo, usado conjuntamente con el planificador difuso, genera información sobre la calidad de las posibles acciones del planificador. Usando esta información, la red neuronal puede ser mejorada. Un diagrama se muestra en la figura 1:

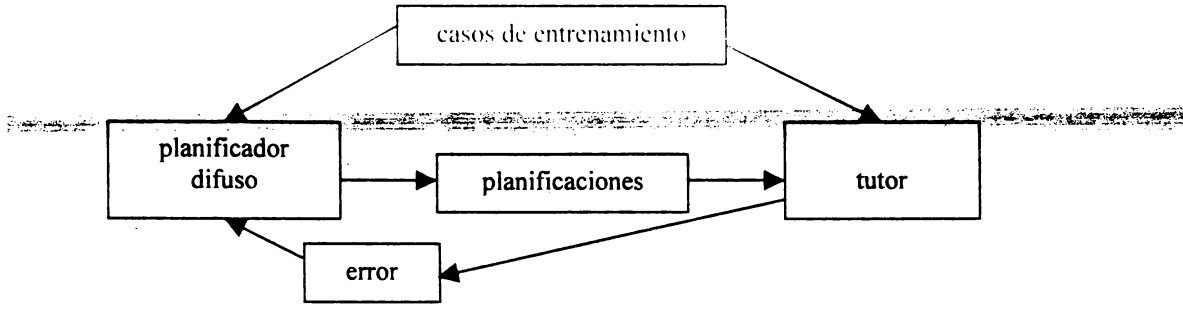



Figura 1: diagrama del proceso de auto-adaptativo

### 3.1 La red neuronal

La figura 2 muestra la red neuronal equivalente al sistema difuso definido anteriormente. Esta red es alimentada con los valores de cada uno de los atributos definidos por el usuario y obtiene la prioridad a la salida. Para mas detalles sobre su construcción consultar las referencias. Una vez conocida la salida esperada, el error es fácilmente corregido a través de la actualización de las funciones de pertenencia.

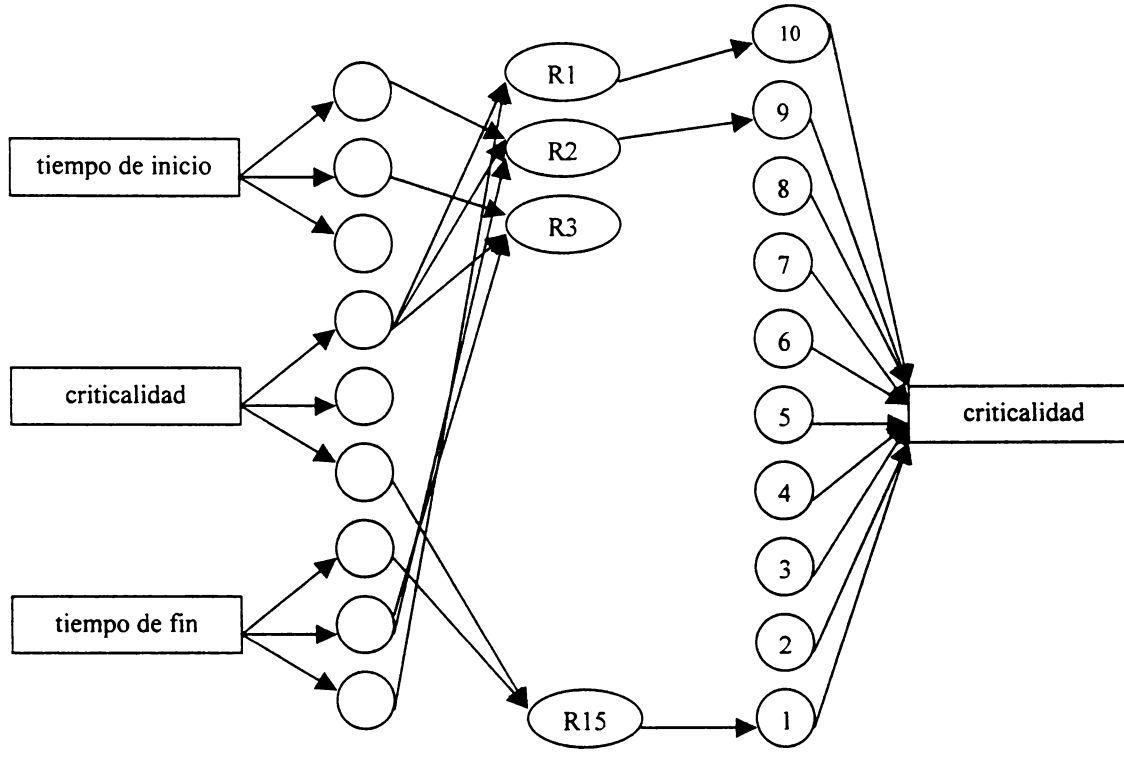



Figura 2: red neuronal equivalente al sistema difuso

### 3.2 El tutor

Se puede considerar al problema de planificación como un problema de aprendizaje por refuerzo, donde el planificador es el agente que toma decisiones en cada estado. En cada paso, el agente recibe una representación del ambiente (valores de los atributos) y selecciona una acción de un conjunto de acciones posibles (elige una prioridad). En etapas posteriores recibe una recompensa. Su objetivo es tomar las decisiones que le permiten maximizar la recompensa, o en otras palabras, lograr las mejores planificaciones.

Un algoritmo de Q-learning se aplica para maximizar la recompensa acumulada a largo plazo. Para evitar el análisis de un espacio muy grande de estados, se agrega al algoritmo de Q-learning el planificador difuso. El planificador es usado para obtener una salida (prioridad) y con una perturbación se le permite explorar estados vecinos.

## 4. Un ejemplo

Dadas dos hebras definidas con los siguientes atributos:

| identificador | inicio | criticalidad | finalización |
|---|---|---|---|
| A | 0 | 10 | 90 |
| B | 10 | 5 | 105 |

El planificador encuentra una planificación que permite cumplir con los tiempos de finalización, tal como se muestra en la figura 3, dado que asigna a A una prioridad de 9, y a B una prioridad menor de 8.

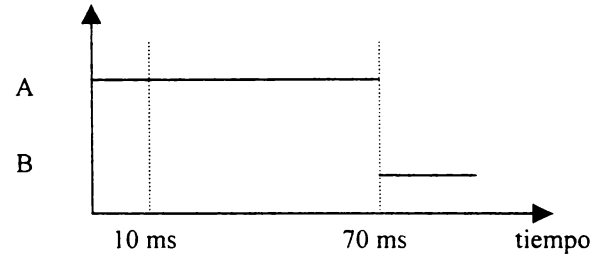



Figura 3: planificación exitosa

Con el objeto de evaluar el funcionamiento del algoritmo auto-adaptativo propuesto, se modifica intencionalmente un conjunto difuso para forzar al planificador a tomar decisiones erróneas. Se selecciona el conjunto de salida 'ocho' y se modifica sus parámetros para hacerlo más parecido a nueve, lo que producirá un error en la asignación de prioridades. Los valores elegidos son 8.7 para el centro y 1 y 0.5 para las amplitudes izquierda y derecha respectivamente.

En este caso, las dos hebras reciben el valor nueve como prioridad. Teniendo ambas el mismo valor, ambas hebras permanecen en la misma cola de prioridad recibiendo cada una

un quantum de ejecución. La tarea A es demorada lo suficiente como para perder su deadline, como se muestra en la figura 4.

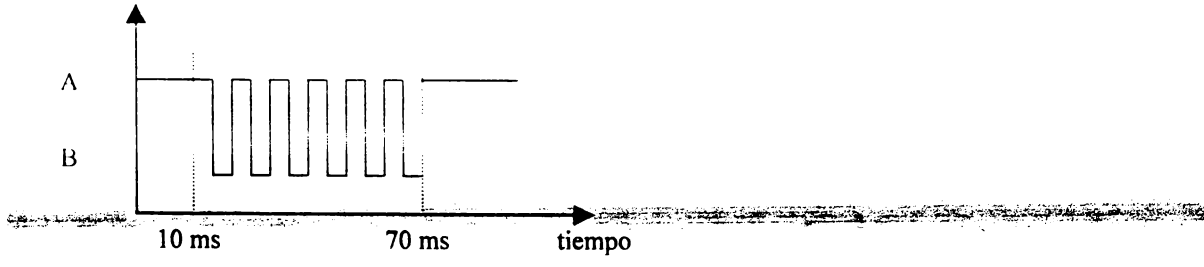



Figura 4: planificación no exitosa.

El diagrama de estados con los valores obtenidos por el algoritmo de Q-learning se muestran en la figura 5. El camino adecuado seleccionado por el tutor se encuentra remarcado.

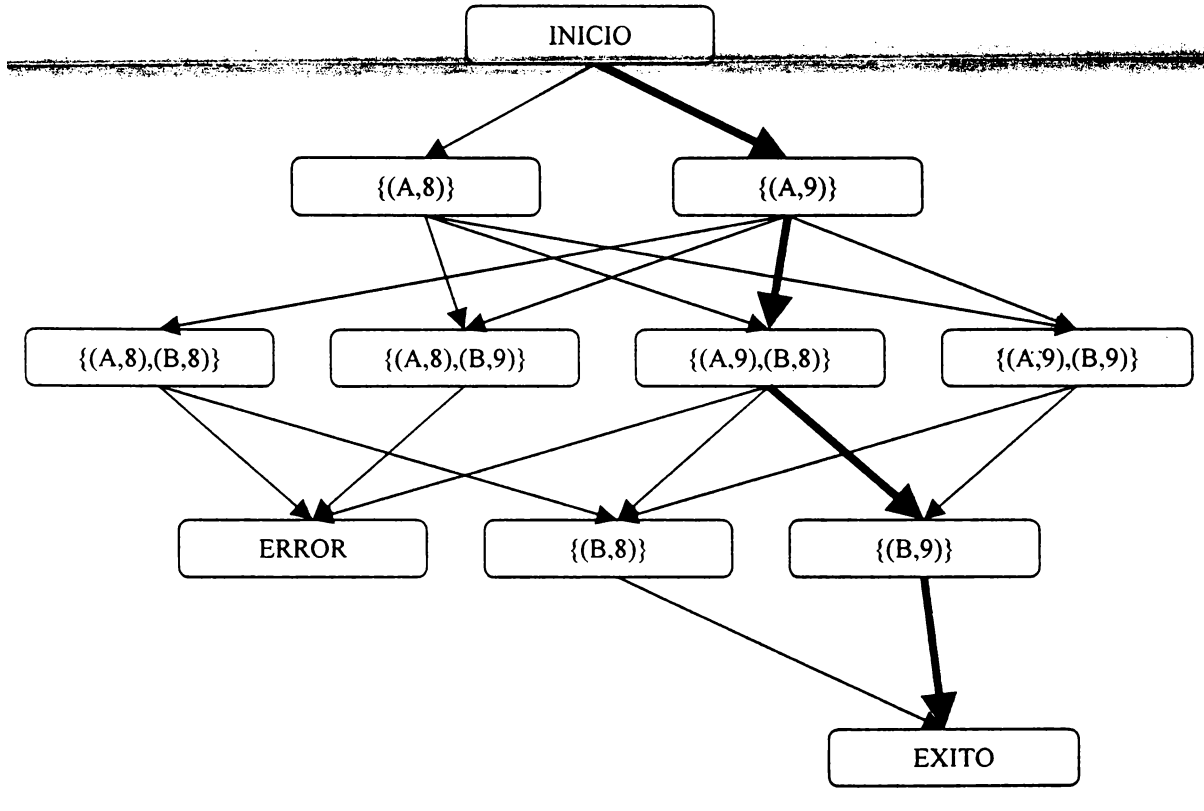
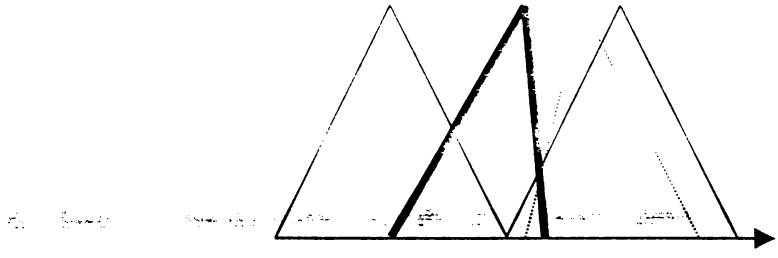



Figura 5: Grafo de estados

El siguiente paso consiste en la utilización de la red neuronal para corregir el error en los conjuntos difusos. Luego del entrenamiento basado en el camino obtenido como adecuado, el error que se introdujo en el conjunto difuso 'ocho' queda corregido. Sus nuevos valores son 8.2 para el centro, 1.3 para la amplitud izquierda y 0.15 para la amplitud derecha. La figura 7 muestra el efecto del aprendizaje. El conjunto dibujado con línea de puntos es el

conjunto modificado para que produzca errores. El conjunto dibujado con línea gruesa el obtenido luego del entrenamiento.



## 5. Conclusiones

Se ha propuesto un nuevo mecanismo a través del cual se pueden definir algoritmos de planificación para hebras, a través de reglas difusas, considerando atributos definidos por el usuario.

Se ha propuesto un algoritmo de auto-adaptación para mejorar el desempeño de los planificadores. Combina una aproximación de redes neuronales con un algoritmo de aprendizaje por refuerzo. El espacio de estados se reduce usando la salida del planificador difuso como guía en el proceso de aprendizaje.

Los algoritmos se encuentran implementados en una biblioteca de hebras Posix para Linux.

## Bibliografía


[1] R. S. Sutton and A. G. Barto (1998). An Introduction to Reinforcement Learning.

[2] J. Yen, J. Lee, N. Pfluger and S. Natarajan (1992). Designing a Fuzzy Scheduler for Hard Real Time Systems. North American Fuzzy Logic Processing Society (NAFIPS '92). NASA.

[3] T. M. Mitchell (1997). Machine Learning. McGraw Hill.

[4] D. Finkelstein, N. C. Hutchinson, D. J. Makaroff, R. Mechler and G. W. Neufeld. Real Time Hebras Interface. Report from the Department of Computer Science, University of British Columbia, Vancouver, Canada.

[5] H. R. Berenji and P. Khedkar (1992). Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. IEEE Transactions on Neural Networks, vol. 3, no. 5.

[6] A. G. Barto, R. S. Sutton and C. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13:834-846.

[7] B. Nichols, D. Buttlar and J. P. Farrel (1996). Phebras Programming. O'Reilly and Associates, Inc.

# THE MCPC EVOLUTION IN EVOLUTIONARY COMPUTATION

**Esquivel S. C., Leiva H. A., Gallard R. H.**
Proyecto UNSL-338403[1]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis, Argentina.
E-mail: {esquivel, aleiva, rgallard}@unsl.edu.ar
Phone: + 54 2652 420823
Fax : +54 2652 430224

## Abstract

The alternative proposed and known as MCPC [6] has improved the performance of the original Holland's Genetic Algorithms in order to obtain good quality solutions in less execution time. But, it's brought out an important concept in Evolutionary Computation: the relationship between exploration and exploitation of solutions in a problem space. The current presentation briefly describes the improvements of the MCPC approach to face the balance between exploration and exploitation.

## 1. Introduction

Important research has been made in evolutionary computation, to maintain a good balance between exploration and exploitation of solutions in a problem space. This research has involved the study of the effect of selection mechanisms. Strong selective pressure can lead to premature convergence towards local optima, while the opposite can make the search ineffective because it would take long time to reach some individual near optimum [11].

Also, the recombination stage of an Evolutionary Algorithms (EA) has its own contribution to the search process. A low recombination rate can impede binary schema processing permitting super-individuals to cope the population and leading to premature convergence. On the other hand, a high rate can be, in some cases, too disruptive allowing the loss of good genetic material, slowing and spoiling the search.

The intuition behind the applicability of the crossover operator is information exchange between different potential solutions. The common approach is to operate once on each mating pair after selection. There, we call such a procedure SCPC (*Single Crossover Per Couple*) approach. Mimicking, in some degree, what happens in nature we devised a different approach to allow multiple offspring per couple and called it MCPC (*Multiple Crossovers per Couple*)[6].

MCPC tries to promote the exploitation of good previously found solutions favouring, with more copies, the best individuals in the current population. We achieved this by repeatedly applying the crossover method to the selected mating pairs; so each of them generates multiple children.

The idea of multiple children per couple was tested on a set of well-known testing functions (De Jong functions $F_1$, $F_2$ and $F_3$[1], Schaffer $F_6$ [12] and other functions). A simple genetic algorithm, with conventional operators and parameter values, was the basis of those initial experiments. Allowing multiple crossing between selected parents similar and better quality solutions were obtained when contrasted against the conventional crossover approach (SCPC). Also, a deeper

[1]The Research Group is supported by the Universidad Nacional de San Luis and the CONICET (Science and Technology Research Council).