# Balanceo óptimo de la carga de artículos en máquinas inyectoras monocolor, usando Algoritmos Genéticos

Ing. Raúl Oscar Klenzi, Gustavo Fernández
Departamento e Instituto de Informática (LISI/IdeI).
Facultad Ciencias Exactas Físicas y Naturales.
Universidad Nacional de San Juan.
rklenzi@iinfo.unsj.edu.ar

## RESUMEN

En el presente trabajo se pretende aplicar una alternativa derivada de la Inteligencia Artificial que son los Algoritmos Genéticos excelente metodología de exploración y explotación de grandes espacios de búsquedas. En éste caso concreto se los utilizará como optimizadores, característica que los destaca sobre otras estrategias de optimización.

Se pretende minimizar la diferencia entre puntas de cuatro máquinas inyectoras de suelas en una fábrica de zapatillas, tarea ésta, que al momento, se realiza a mano apoyado en tablas históricas y el presente trabajo ha permitido encontrar respuestas óptimas en mucho menos tiempo.

Para la aplicación se utiliza una herramienta disponible en Internet desde la cual se manipulan los algoritmos en sí, en cuanto a estrategias de selección, transposición, mutación y de terminación; a lo que debe adicionarse una correcta elección de la representación y función de aptitud.

# 1. INTRODUCCION

El trabajo que aquí se expone tiene como objetivo encontrar una distribución de artículos en máquinas inyectoras monocolor de modo que éstas estén balanceadas, es decir que cada una de ellas tenga una cantidad similar en su cola de artículos pendiente de inyección.

La idea de hacer este estudio surge motivado, que con la actual distribución, normalmente se observa un notable desbalanceo en la carga de máquinas.

La empresa para la que se desarrolla la aplicación, cuenta con 4 inyectoras monocolor, todas de iguales características. Para cada artículo posee un molde por número de cada grupo de artículos. Por ejemplo: al grupo de los 1500 lo integran el 1506, 1509, 1512, 1516 y el 1555. Los moldes pueden colocarse en cualquiera de las máquinas. Al tener un solo molde por número dos máquinas no pueden inyectar el mismo número del mismo artículo al mismo tiempo. Cuando se comienza con la inyección de un artículo, ésta se hace en todos sus números, de modo que no se permite la mezcla de artículos. Sólo entrará a inyección un nuevo artículo cuando la máquina inyectora haya terminado con todos los números que corresponden a dicha máquina del artículo anterior.

Planteado el presente problema proponemos dos alternativas para solucionarlo.

## 1.1 Primer Alternativa de Solución

Se parte del análisis de los pedidos de los distintos artículos que fabrica la empresa, en el periodo abarcado por los años 1997-1998.

Los datos se relevaron de los Cuadernos de Pedidos de Corte, de los cuales se deriva su posterior aparado e inyección.

Para cada uno de los artículos se fueron insertando en una tabla los distintos pedidos que se realizaron, previamente clasificados en artículos de Niño, Dama y Hombre.

Posteriormente se calculó un promedio histórico de los pedidos de cada uno de los artículos.

A continuación se obtuvo una curva característica de cada grupo de artículos, por ejemplo, una para los artículos 100, otra para los 300 y otra para los 500, y así sucesivamente.

Basados en estos pedidos típicos encontrados para cada grupo de artículos se distribuyeron en las máquinas inyectoras de acuerdo a una cantidad de pares que mantuviera un balance de la carga de las mismas.

## 1.2 Segunda Alternativa de Solución

Esta propuesta de solución es mediante el uso de algoritmos genéticos.

### Optimización usando Algoritmos Genéticos

Los Algoritmos genéticos (AG) son técnicas que resuelven problemas de optimización inspirados en la teoría de la evolución y la biogenética. Son una buena

alternativa explorando grandes espacios de búsqueda donde existen óptimos globales y locales. Los aspectos básico de los AG son :

1. Representan las posibles soluciones del problema como un string de parámetros (números). Este string es llamado ***cromosoma*** y los parámetros ***genes.***
2. Aleatoriamente crean un número (generación) de estos cromosomas.
3. Se calcula la aptitud de cada cromosoma como forma de clasificar los mismos en orden a su función de aptitud, seleccionando los más aptos.
4. Se crea una nueva generación de cromosomas, seleccionando al azar un par de cromosomas (padres) y mezclando sus genes se logran los cromosomas hijos. Este proceso es llamado ***crossover*** y la selección de los padres se hace de acuerdo a la función de aptitud de cada uno.
5. Repitiendo los pasos 3 y 4 para un número dado de ciclos (generaciones).

La aleatoriedad del proceso anterior permite la efectiva exploración de un gran espacio de soluciones. Mientras la selección de soluciones efectivas (cromosomas) y la mezcla de sus genes permite la acumulación de buenas características desde soluciones parcialmente buenas. Como resultado, los AG exploran grandes dominios y convergen a buenas soluciones relativamente rápido. Dando una buena relación entre el tiempo empleado y la calidad de la solución. Típicamente se desea encontrar una buena solución en un corto tiempo y no la solución óptima en un tiempo excesivo.

Los dos pasos básicos en la búsqueda de soluciones usando AG son: una adecuada representación del problema y un método para encontrar la función de aptitud (función de costo) a evaluar permanentemente. La representación más sencilla de un problema es por medio de un string de números. Cada número se representa por un gen que puede estar restringido a una valor máximo y mínimo. La función de costo se define como el costo de una solución dada para un conjunto de valores de genes. En tal representación cada gen representa un parámetro diferente de la solución.

## 2. EL PROBLEMA

El trabajo que aquí se expone, adelantado ya en la introducción, tiene como objetivo encontrar una distribución de artículos en máquinas inyectoras monocolor de modo que éstas estén balanceadas, es decir que cada una de ellas tenga una cantidad similar en su cola de artículos pendiente de inyección. Significa que entre las máquinas que más y menos suelas inyecten respectivamente, la diferencia de cantidades se minimice.

## 3. HERRAMIENTA UTILIZADA

Se ha definido ya el uso de AG como metodología de búsqueda de una solución, para ello se hará uso de una herramienta del Instituto de Tecnología de Massachusset MIT, disponible en INTERNET y que no es más que una colección de bibliotecas en C++ que permiten generar y correr distintos tipos de Algoritmos Genéticos.

La herramienta aludida recibe el nombre de GALIB2.42 y absolutamente todas sus características se encuentran en el correspondiente archivo HTML y en el presente informe solo destacaremos algunas de ellas.
Hay tres cosas que se deben analizar al intentar resolver un problema mediante Algoritmos Genéticos.

1. Definir una representación.
2. Definir el operador genético.
3. Definir la función objetivo.

GALIB ayuda con los dos primeros ítems proveyendo ejemplos desde los cuales se pueden construir la representación y operadores particulares. En muchos casos se pueden usar las representaciones y operadores existentes con pequeñas modificaciones. La función objetivo se desarrolla completamente por el usuario de GALIB. Una vez definidas la representación, operadores y una medida del objetivo, se puede aplicar cualquier Algoritmo Genético para encontrar la mejor solución a un problema dado.

Cuando se usa un Algoritmo Genético para resolver un problema de optimización, se debe estar en condiciones de representar una solución a l problema en una única estructura de datos. El Algoritmo Genético (AG) creará una población de soluciones basada en una muestra de la estructura de datos provista. El AG opera sobre la población y evoluciona hacia la mejor solución. En GALIB la librería contiene cuatro tipos de Genomas:

- GAListGenome.
- GATreeGenome.
- GAArrayGenome.
- GABinaryStringGenome.

Clases que se derivan de la clase base GAGenome y una estructura de datos como la indicada por sus nombres como ejemplo: GAListGenome se deriva de la clase GAList también como de la clase GAGenome. Usa una estructura de datos que trabaja con la definición del problema a resolver.
Hay diferentes tipos de Algoritmos Genéticos. Galib incluye cuatro tipos básicos:

- Simple, ***GASimpleGA ga(genome);***
- Estado-Permanente, ***GASteadyStateGA ga(genome);***
- Incremental, ***GAIncrementalGA ga(genome);***
- Islas de poblaciones, ***GADemeGA ga(genome);***

Estos algoritmos difieren en la forma en que crean nuevos individuos y reemplazan los viejos durante el curso de la evolución.

GALIB provee dos mecanismos primarios para extender las capacidades de construir objetos. Sobre todo (y desde el punto de vista del C++) se pueden derivar sus propias clases y definir nuevas funciones miembros. Solamente se necesitan hacer ajustes menores al comportamiento de la clase Galib, en la mayoría de los casos se define una sola función y se le dice a la clase Galib que la use en lugar de las definidas por default.

Los AG cuando están implementados apropiadamente, son capaces de realizar tanto la tarea de exploración (búsqueda a lo ancho) como de explotación (búsqueda local) del espacio de búsqueda. El comportamiento que tendrá dependerá como trabajen los operadores, y de la forma del espacio de búsqueda.

Como anteriormente se mencionó un ciclo de un AG involucra tres operaciones, selección, transposición y mutación. GALIB posee distintas alternativas para cada uno de estos operadores:

## 3.1 Selección

- *GARouletteWheelSelector().*
- *GARankSelector().*
- *GATournamentSelector().*
- ***GADSSelector().***
- *GASRSSelector().*
- *GAUniformSelector().*

## 3.2 Transposición

- ***ga.crossover(GARealPartialMatchCrossover).***
- *ga.crossover(GARealTwoPointCrossover).*
- *ga.crossover(GARealGenome::UniformCrossover).*
- *ga.crossover(GARealGenome::EvenOddCrossover).*
- *ga.crossover(GARealGenome::OnePointCrossover).*
- *ga.crossover(GARealGenome::OrderCrossover).*
- *ga.crossover(GARealGenome::CycleCrossover).*

## 3.3 Mutación

- ***genome.mutator(GARealSwapMutator).***
- *genome.mutator(GARealFlipMutator).*
- *genome.mutator(GARealGaussianMutator).*

Se sugiere que para más información sobre las características del soft, remitirse a: mbwall@mit.edu.

Las funciones elegidas son las remarcadas y subrayadas, evolucionando con un algoritmo genético de población en islas, con probabilidad de mutación 0.0011 y de crossover 0.999 y estrategia de terminación de las generaciones basada en la convergencia durante un cierto numero de generaciones.

## 3.4 Definiendo una Representación

Para el presente problema se opto por la representación ***GARealGenome*** que es una especialización de la estructura ***GAArrayGenome***, de la librería Galib. Aquí la representación es un string de números reales.
A continuación se considera un ejemplo de la forma que tendría esta representación.

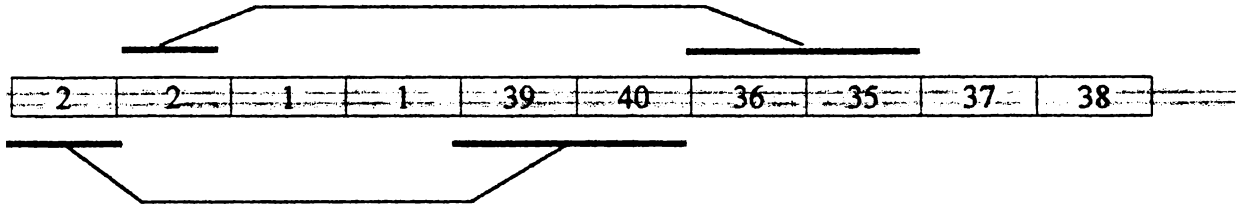
**Ejemplo:**

Supongamos el caso de un pedido del artículo 301, el cual se fabrica en seis números diferentes (35,36,37,38,39,40) aquí el string (genoma-cromosoma) tendría una longitud equivalente a la cantidad de maquinas inyectoras (4) más la cantidad de números distintos en los que se fabrica dicho producto(6). Así el cromosoma es de longitud 10.

| Gen1 | Gen2 | Gen3 | Gen4 | Gen5 | Gen6 | Gen7 | Gen8 | Gen9 | Gen10 |
|---|---|---|---|---|---|---|---|---|---|

Las posiciones desde el Gen1 hasta el Gen 4 se corresponden con la cantidad de números diferentes del artículo que se inyectan en cada máquina. Las posiciones restantes, desde Gen 5 hasta Gen 10 son ocupadas por los números de artículo en el orden en que se distribuyen en las máquinas .

Si tenemos el siguiente string:




Nos indica lo siguiente:

La máquina 1 deberá inyectar dos números diferentes: 39 -40

La máquina 2 deberá inyectar dos números diferentes: 36- 35

La máquina 3 deberá inyectar un solo número: 37

La máquina 4 deberá inyectar un solo número: 38

En el ejemplo tratado, la cantidad de strings que pueden llegar a generarse, y que conforman el espacio de búsqueda, es igual a:

**ESPACIO DE BUSQUEDA** $= 3^4 * 6^6 = 3.779.136$

$3^4$ Debido a que las 4 primeras posiciones del string desde el Gen1 hasta el Gen 4 deben ser distintas de 0 (ninguna de las máquinas puede quedar sin un número para inyectar), estar entre 1 y 3 (pues si alguna de estas posiciones tomara un valor mayor que 3 otra debería tomar el valor 0, ya que el resultado de la suma de las cuatro posiciones debe ser 6, pues son 6 números diferentes del mismo artículo los que se deben distribuir en las cuatro máquinas.

$6^6$ Ya que son 6 números comprendidos entre 35 y 40 los que deben ocupar las seis posiciones restantes del genoma desde la posición Gen 5 hasta la posición Gen 10

**RESPUETAS CORRECTAS** = 24 * 6 =144.

Es tan válida una respuesta,

| 1 | 1 | 1 | 3 | 36 | 37 | 38 | 39 | 40 | 35 |
|---|---|---|---|---|---|---|---|---|---|

como otra...

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 1 | 1 | 40 | 35 | 39 | 36 | 37 | 38 |


Esto significa que la exploración, para encontrar la o las respuestas correctas, se debe hacer sobre 3.779136 strings distintos entre los que se encuentran algunos como:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 1 | 1 | 39 | 38 | 40 | 35 | 37 | 40 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 1 | 2 | 39 | 35 | 39 | 37 | 40 | 38 |

Y que deberán ser eliminados, por el algoritmo utilizado, atendiendo a las restricciones que se estén considerando. De hecho la restricción de que todos los números inyectados sean distintos eliminaría los dos últimos strings presentados.
Para reflejar un poco más la relación entre el espacio solución y el espacio de exploración, se expone el caso de considerar otro producto que procesa 16 números distintos. En este caso el string dispondrá de 20 genes; los 4 primeros tienen el mismo significado que en el caso anterior, los 16 restantes son los números distintos a inyectar.
El espacio a explorar es de $5.268574574892*10^{23}$ y las posibles soluciones son: 7.962.624, es decir una solución entre $6.61663119204*10^{16}$.

## 3.5 Definiendo la función de Aptitud

En el presente ejemplo la función de aptitud verifica que se cumplan las siguientes restricciones:

- Las cuatro primeras posiciones del string deben sumar la cantidad de números que determinado artículo fabrica y ser distintas de cero.
- Se debe verificar que desde la quinta y hasta la última posición del string los números sean distintos y estén comprendidos entre el menor y el máximo número a inyectar.
- Como cada número a inyectar tiene asociada una cantidad a producir, la aptitud de cada string se evalua mediante la diferencia entre la máquina que más cantidad de pares elabora y la que menos produce, para los distintos string.

## 3.6 Característica de la Población Inicial

La poblacion inicial se genera en un 50% en forma aleatoria y en el restante 50% con una heuristica basada en lo siguiente:
Se ordenan de mayor a menor las cantidades a elaborar de cada número
De acuerdo al ejemplo de seis números.

**Número 36 cantidad a producir 120**
Número 37 cantidad a producir 120
Número 38 cantidad a producir 90
Número 39 cantidad a producir 48
Número 35 cantidad a producir 30
Número 40 cantidad a producir 24

Luego se generan los cuatro primeros genes de la representación en forma aleatoria y se ordenan de menor a mayor. Los genes siguientes se conforman atendiendo a tomar alternativamente los extremos superior e inferior de las cantidades hasta agotar el gen correspondiente

## 4. SOLUCIONES ENCONTRADAS

A continuación se muestra, a través de un ejemplo (pedido promedio de artículos 500), las diferencias entre la distribución actual de números en c/u de la máquinas inyectoras, la distribución según la primer solución propuesta y la distribución de acuerdo con la segunda solución propuesta elaborada con AG.

**PEDIDO PROMEDIO DE ARTICULOS 500**

| **Numero** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Cantidad** | 228 | 317 | 381 | 392 | 417 | 551 | 592 | 592 | 472 | 346 | 242 |

**DISTRIBUCION ACTUAL**
**Artículo 500**

| **Maquina 1** | **Máquina 2** | **Máquina 3** | **Máquina 4** |
|---|---|---|---|
| 38 – 40 | 37 – 41 – 45 | 36 – 42 – 44 | 35 – 39 – 43 |
| 943 pares | 1215 pares | 1255 pares | 1117 pares |

**Diferencia entre máquinas: M3 – M1= 312 pares* 1.22 minutos = 6.34hs**

DISTRIBUCION SEGUN LA PRIMER SOLUCION PROPUESTA

## Artículo 500

| Maquina 1 | Máquina 2 | Máquina 3 | Máquina 4 |
|---|---|---|---|
| 35 - 36 - 44 - 45 | 37 - 38 - 39 | 40 - 41 | 42 - 43 |
| 1133 pares | 1190 pares | 1143 pares | 1064 pares |

**Diferencia entre máquinas: M2 – M4= 126 pares * 1.22 = 2.56 hs.**

## DISTRIBUCION SEGÚN LA SEGUNDA DISTRIBUCION PROPUESTA
## Artículo 500

| Maquina 1 | Máquina 2 | Máquina 3 | Máquina 4 |
|---|---|---|---|
| 37 – 38 – 44 | 35 – 36 – 41 | 40 – 42 | 39 – 43 – 45 |
| 1119 pares | 1137 pares | 1143 pares | 1131 pares |

**Diferencia entre máquinas: M3 – M1= 24 pares * 1.22 minutos = 0.48 hs.**

## 5. CONCLUSIONES DE LA SOLUCIÓN ENCONTRADA MEDIANTE ALGORITMOS GENÉTICOS.

- Converge a la solución óptima, que garantiza la mínima diferencia entre puntas. Significa que entre la máquina que más produce y la que menor cantidad de inyecciones realiza, existe la menor diferencia, que se traduce en ahorro de tiempo y energía.
- La respuesta óptima se encuentra en forma automática, mediante el soft y no en forma manual.
- El tiempo de cálculo es, en promedio para las distintas soluciones, de 10 segundos en una PC con procesador Pentium 166Mhz y 32 Mbytes en RAM.

## 6. BIBIOGRAFÍA CONSIDERADA


- *Genetic Algorithms in Search and Optimization,* **David Edward Goldberg, Addison-Wesley Pub. Co., 1989 ISBN 0-201-15767-5.**
- ***Un criterio para seleccionar operadores genéticos para resolver CSP. Trabajo presentado en el CACIC '98.(paper)***
- ***Heuristic Genetic Algorithms for Constrained Problems A.E. Eiben, P-E. Raué, Zs. Ruttkay. Artificial Intelligence Group. Departament of Mathematics and Computer Science Vrije Universiteit Amsterdam.(paper)***
- ***Solving Constrain Satisfaction Problems Using Genetic Algorithms. A.E. Eiben, P-E. Raué, Zs. Ruttkay. Artificial Intelligence Group. Departament of Mathematics and Computer Science Vrije Universiteit Amsterdam.(paper)***
- ***A Genetic Algorithm for Resource_Constrained Scheduling*** Tésis Doctoral de Matthew Bartschi Wall, Massachusetts Institute of Tecnology .