# Una metaheurística híbrida auto-adaptativa multi-capa para la resolución simultánea de múltiples instancias[1]

A.D. Masegosa, A. Sancho Royo, D. Pelta
Grupo de Modelos de Decisión y Optimización
Departamento de Ciencias de la Computación e IA, Universidad de Granada
admase@decsai.ugr.es, alejandrosanchoroyo@gmail.com, dpelta@decsai.ugr.es

## Resumen

La mayoría de las metaheurísticas con mecanismos de auto-adaptación que se han propuesto en la literatura, afrontan cada resolución desde el principio, sin tener en cuenta lo aprendido durante otras búsquedas. Basándonos en la idea de que el esfuerzo computacional realizado para la resolución de una instancia debería ser aprovechado en la solución de instancias similares, presentamos un nuevo modelo que permite la solución simultánea de un grupo de instancias. Este modelo contiene un conjunto de agentes que operan sobre un espacio de búsqueda. Presenta un esquema multi-nivel de 2 capas que controla jerárquicamente la auto-adaptación de los parámetros de los operadores. Este nuevo método ha sido aplicado al problema MAX-SAT con conjuntos de varias instancias obteniéndose unos resultados prometedores.

## 1. Introducción

En el contexto de los problemas de optimización, las metaheurísticas se han mostrado como las técnicas no exactas más eficientes, especialmente si pretendemos encontrar herramientas de propósito general. Una revisión en castellano sobre metaheurísticas puede encontrarse en [5]. La mayoría de estas técnicas que han sido ampliamente estudiadas abordan la solución de una instancia de un problema de optimización, pero una vez obtenida una solución -óptima o, cuando menos de una calidad aceptable, en dependencia del problema- el proceso de resolución de una nueva instancia debe ser abordado desde el principio.

Basándonos en la idea de que el esfuerzo computacional realizado para la resolución de una instancia debería ser aprovechado en la solución de instancias similares, presentamos una nueva metaheurística.

Este modelo pretende tener un alto nivel de autoajuste y está orientado a ser de propósito general.

En la Sección 2 presentamos formalmente el modelo destacando sus elementos y los sistemas de autoajuste .

En la sección 3 presentamos el problema elegido como banco de pruebas –en una primera aproximación- de este modelo. El MAX-SAT es un problema clásico, bien estudiado, del que pueden extraerse fácilmente instancias de dificultad variable y que puede ser considerado un banco de pruebas accesible y fácilmente contrastable por cualquier otro investigador.

La Sección 4 está dedicada a presentar los experimentos realizados y las conclusiones.

## 2. Modelo

El modelo que exponemos en este trabajo se estructura en dos capas, como se puede observar en la Figura 1. La primera está compuesta por una población de Agentes de Modificación de Soluciones (AMS) que actúan sobre un conjunto de soluciones para evolucionarlas mediante la aplicación de unos operadores que tienen asociados. El comportamiento de estos operadores se puede variar durante la búsqueda a través de ciertos parámetros sobre los que actúa la segunda

[1] Este trabajo ha sido parcialmente financiado por el programa de becas FPI bajo el proyecto TIN-2005-08404-C04-01 y por TIC-00129-PE

capa. Esta capa consta de una población de Agentes de Modificación de Parámetros (AMP) que asignando diferentes valores a tales parámetros intenta optimizar el funcionamiento de los operadores.

Una novedad importante que incorpora la metaheurística presentada es la posibilidad de resolver de forma simultánea un conjunto de instancias. En lo que resta de apartado exponemos la descripción formal del modelo.

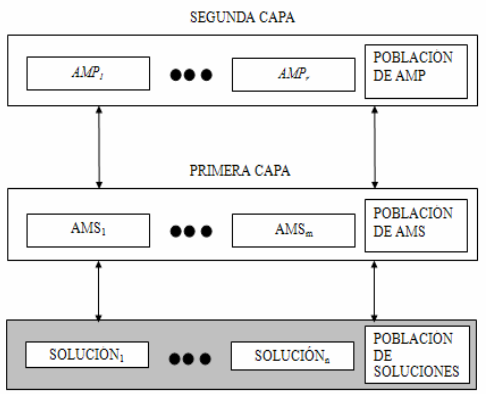


Figura 1.

Consideremos un problema de optimización. El espacio de soluciones factibles del problema es $S$ y $\mathbb{F}$ un conjunto de funciones objetivo $f : S \rightarrow \mathbb{R}$ de un grupo de instancias de este problema para las que supondremos, sin perdida de generalidad, que estamos buscando un mínimo. De aquí en adelante consideraremos una solución $s$ como la tupla $(x, f)$ con $x \in S, f \in \mathbb{F}$.

Por otra parte, tenemos un conjunto de operadores $\Theta = \{O_1, .., O_m\}$ donde $O : S \times P \rightarrow S$ obtiene una nueva solución $s_{i+1} = (x_{i+1}, f)$ a partir de $s_i = (x_i, f)$. Cada aplicación del operador sobre la misma solución $s_i$ devuelve resultados distintos, esto es, tiene algún factor de aleatoriedad. El comportamiento de un operador puede ser modificado mediante un vector de parámetros $t = (t_1, ..., t_l) \in P$ que toma valores en el dominio $D = D_1 \times ... \times D_l, P \subseteq D$.

Un Agente de Modificación de Soluciones viene definido por la tupla:

$$AMS(O, t, f_{acept})$$

$O$ es un operador del espacio de operadores $\Theta$ y $f_{acept} : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^n \rightarrow \{true, false\}$ un determinado criterio de aceptación. Este criterio puede ser estocástico (temple simulado), tabú, difuso[6], etc. Un *AMS* recibe una solución $s_i$ como entrada y a partir de ella obtiene $s_{i+1} = O(s_i, t)$. Entonces decide dar como salida $s_i$ o $s_{i+1}$ en función de $f_{acept}(f(s_i), f(s_{i+1}), r)$, donde $r \in \mathbb{R}^n$ es una tupla de parámetros de control. Cada vez que este proceso tiene lugar el AMS incrementa/decrementa su nivel de energía en una determinada cantidad que viene establecida por:

$$\Delta energía = \begin{cases} f(s_i) - f(s_{i+1}) \ si \ f(s_i) - f(s_{i+1}) > 0 \\ 0 \ en \ otro \ caso \end{cases}$$

Un Agente de Modificación de Parámetros se define por la tupla:

$$AMP(t_i, O, V)$$

con $V \subseteq D_i$. La función de un $AMP$ es ajustar el comportamiento de los $AMS$ que tienen asociado el operador $O$, modificando la componente i-ésima del vector de parámetros $t$ mediante la asignación de un valor $v \in V$. La elección se hace en base a un vector de pesos $\omega = (\omega_1, ..., \omega_s), \omega \in \mathbb{R}^s$ que asocia a cada elemento de V un número real. Concretamente, se calcula para cada $v \in V$ una probabilidad de ser seleccionado:

$$P(v) = \omega(v) / \sum_{x \in V} \omega(x)$$

Una vez obtenidas estas probabilidades, se escoge un valor mediante el método de la ruleta.

Comenzando ya con la descripción de la célula tenemos que decir que esta se estructura jerárquicamente en dos capas.

La primera capa contiene una población de soluciones $Pob = \{s_1, ..., s_n\}$. Denotaremos por $Pob(f) = \{s \in Pob \mid s = (x, f)\}$ a la subpoblación de

los soluciones que pertenecen a una misma instancia. Se cumple que $|Pob(f)|$ es el mismo $\forall f \in \mathbb{F}$. Por otro lado, tenemos una población de $AMS$, $A1 = \{AMS_1, \ldots, AMS_m\}$.En cada iteración de esta capa, en primer lugar se genera una permutación aleatoria de *A1* con el fin de aplicar los *AMS* en un orden distinto cada ciclo. Tras esto, tiene lugar el siguiente bucle:

**Para cada** $AMS \in A1$
  **Extraer solución** $s_i$ **de** $Pob$
  $\hat{s}_i \leftarrow AMS(s_i)$
  **Reemplazar** $s_i$ **por** $\hat{s}_i$
  $i \leftarrow (i+1) \bmod n$
**Fin Para**

Algoritmo 1.

La función de la segunda capa es ajustar los parámetros de los *AMS* de *A1*. Para ello tenemos una población de $AMP, A2 = \{AMP_1, \ldots, AMP_q\}$. Cada $AMP$ ajusta el parámetro $t_i$ de aquellos $AMS \in A1$ que tienen asociado el operador $O_k \in \Theta$. Este subconjunto de *A1* lo denominaremos $A1^k$.

En una primera fase, se actualizan los vectores de pesos de los agentes de *A2*. Este proceso se realiza de la siguiente manera:

Sea $A1^k(v)$ el subconjunto de $A1^k$ de aquellos agentes a los que se les ha asignado el valor $v \in V$ al parámetro $t_i$ y $energía(a,it), a \in A1$, la energía del agente *a* en la iteración *it*. El peso asociado al valor $v \in V$ en $it, \omega(v,it)$, se calcula como sigue:

$$r(v,it) = \sum_{a \in A^k(v)} \frac{energía(a,it) - energía(a,it-1)}{|A^k(v)|}$$

$$\omega(v,it) = \omega(v,it-1) + r(v,it)$$

Una vez actualizados los pesos, los agentes de *A2* asignan nuevos valores a los parámetros de los agentes de *A1*.

Para finalizar este apartado, y una vez que ya hemos desglosado el funcionamiento de cada una de las capas que componen la metaheurística, vamos a mostrar el algoritmo que describe el funcionamiento global de modelo:

```
Inicialización de la 1ª capa
Inicialización de la 2ª capa
while(¬condición de parada)
   ejecutar 1ª capa
   if(condición 2ª capa)
       ejecutar 2ª capa
   end if
end while
```

Algoritmo 2.

## 3. El problema MAX-SAT

Dentro del campo de la optimización combinatoria, el problema de la satisfacibilidad (SAT) es uno de los más conocidos. Juega un papel muy importante tanto en la teoría de la complejidad como en la inteligencia artificial, con una gran cantidad de aplicaciones prácticas como son el diseño y verificación de componentes hardware, diseño de circuitos asíncronos, diseño de redes de computadores y resolución de problemas de planificación.

En una instancia de SAT tendríamos los siguientes elementos:

- Un conjunto finito de variables booleanas $X = \{x_1, \ldots, x_n\}$ que toman valores en el dominio $B = \{1,0\}$.
- Un conjunto de operadores booleanos $O = \{\wedge, \vee, \neg\}$, siendo $\wedge$ la conjunción, $\vee$ la disyunción y $\neg$ la negación
- Un conjunto finito de literales L, dónde un literal puede ser una variable o su negación. Esto es, $L = \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$.
- Un conjunto finito de cláusulas $C = \{c_1, \ldots, c_m\}$, dónde una cláusula es un conjunto finito de literales conectados por los operadores $\wedge$ o $\vee$.
- Una fórmula booleana *F* formada por un conjunto de cláusulas conectadas por los operadores $\wedge$ o $\vee$. Si se cumple que $F = c_1 \wedge \ldots \wedge c_m$ y que $c_i = l_1 \vee \ldots \vee l_k$, $\forall c_i \in C, l_i \in L$ se dice que *F* está en Forma Normal Conjuntiva (CNF). Puesto que cualquier fórmula booleana puede ser

transformada a esta forma, podemos suponer, sin perder generalidad alguna, que *F* se encuentra en CNF.

Una asignación de verdad para $X$ es un aplicación $a : X \rightarrow \mathrm{B}$ . Una asignación *a* satisface una cláusula $c_k$ si $\exists l \in c_k$ tal que $l = 1$ bajo *a*. Un modelo para *F* es una asignación de verdad que satisface todas las cláusulas de *F*.

El problema SAT consiste en, dado un conjunto de variables booleanas $X = \{x_1,..., x_n\}$ y un conjunto de cláusulas $C = \{c_1,..., c_m\}$, encontrar un modelo para $F = c_1 \wedge \ldots \wedge c_m$ .

MAX-SAT es la variante de optimización de SAT. En este problema, la función a minimizar es el número de cláusulas no satisfechas. Otra familia de problemas dentro de SAT es la denominada k-SAT, en la que todas las cláusulas tienen exactamente k literales. Cuando k = 2 se han encontrado algoritmos que lo resuelven tiempo polinomial, pero para $k \geq 3$ se ha demostrado que es un problema NP-duro.

## 4. Experimentación y Resultados

El propósito del estudio experimental de este trabajo no es el de comparar este esquema con el estado del arte para MAX-SAT sino evaluar el funcionamiento de nuestra metaheurística al resolver varias instancias de forma secuencial o concurrente. Con esto pretendemos contrastar la hipótesis de si podemos aprovechar el esfuerzo computacional empleado para ajustar los pesos con los que trabajan los AMP, en la resolución de otras instancias.

El banco de pruebas utilizado para el análisis, es un conjunto de instancias que pertenecen a la familia Uniform Random 3-SAT y han sido obtenidas del recurso web SATLIB [2]. Fueron generadas en la fase de transición de resolubilidad, que está en aproximadamente 4,26 cláusulas por variable. Este tipo de instancias presentan una gran dificultad ya sea para algoritmos exactos o para heurísticos como se muestra en [1] y [8] .Dentro de este benchmark hemos seleccionado como conjunto de test las instancias de mayor dificultad de la clase uf100-430, todas con 100 variables, 430 cláusulas, y satisfacibles, es decir, el óptimo se encuentra en 0. Para determinar la dificultad, hemos usado como criterio el número medio de flips para alcanzar el óptimo en 100 ejecuciones del algoritmo WalkSAT[7]. Determinamos como fácil la de menor media, media la mediana y difícil la de mayor promedio. Este método para testear la dureza de las instancias ha sido utilizado en otros trabajos como [3].

La implementación del modelo expuesto anteriormente ha sido realizada con el lenguaje de programación Java en la versión 1.6. Para esta tarea también nos hemos ayudado de la librería LiO [4]. Los experimentos han sido ejecutados en un computador con procesador Core 2 Duo 2GHz y con 1 GB de memoria RAM.

Para la experimentación se ha utilizado codificacion binaria. Se ha usado un único operador, *k-BitFlip*, el cual complementa el valor de *k* variables seleccionadas aleatoriamente. El criterio de aceptación para los AMS es el de Boltzman con $T_0 = 4$ . Se ha fijado la población inicial de *AMS* a 20 y la de soluciones a 90. La segunda capa se ejecuta cada 30 iteraciones de la primera. El conjunto de valores posibles para el parámetro k del operador flip se a fijado a $V = [1,4] \in \mathbb{N}$.

En este análisis preliminar, vamos a mostrar los resultados obtenidos en la resolución simultánea de 3, 5 y 8 instancias. Hemos considerados varios casos en función del retardo con el que se insertan las instancias:

- Caso 1: Todas las instancias son iniciadas a la vez.
- Caso 2: Cada instancia es iniciada con un coeficiente de retardo de 0,00025 veces el total de iteraciones de la primera capa
- Caso 3: Igual al anterior pero con un coeficiente de retardo de 0,00075 .
- Caso 4: Considerando un coeficiente de retardo de 0,001.

Con el fin de contrastar la hipótesis comentada anteriormente, también vamos a mostrar los resultados obtenidos cuando la búsqueda se realiza de forma secuencial. Los experimentos han sido lanzados 30 veces sobre cada caso, fijando la condición de parada a 12, 20

y 32 millones de evaluaciones de la función objetivo para 3, 5 y 8 instancias respectivamente. En un primer análisis vamos a estudiar el comportamiento de la metaheurística cuando añadimos instancias a un determinado conjunto, es decir, los de mayor tamaño contienen a los de menor. En la Tabla 1 mostramos el número de ejecuciones exitosas (aquellas en las que se ha llegado al óptimo en todas las instancias). En esta se puede comprobar que el número de experimentos en los que se resuelven todas las instancias es considerablemente mayor para la búsqueda simultánea, obteniéndose diferencias estadísticamente significativas en la mayoría de los casos. En la Tabla 2 exponemos el porcentaje medio de evaluaciones para sobre el total que el algoritmo ha empleado en las ejecuciones consideradas como exitosas. Se observa que los porcentajes son similares, no encontrándose diferencias significativas entre la resolución secuencial y la paralela, salvo para el Caso 3 con 3 instancias. Esto nos indica que para un esfuerzo computacional similar, la resolución simultánea de instancias nos permite obtener unos resultados más robustos.

Otro aspecto a tener en cuenta son los casos no exitosos, para observar cuantas instancias se quedaron sin resolver. En las figuras 2, 3 y 4 se muestran unos diagramas apilados. La longitud de las barras señala el número de ejecuciones, siendo éstas de tres tipos; de error igual a 0, 1 y 2. Este estudio nos muestra que el caso secuencial presenta un peor comportamiento frente los paralelos a lo largo de los distintos grupos de instancias, ya que hay un mayor porcentaje de ejecuciones en las que el número de instancias sin resolver es 2.

En un segundo bloque de experimentos, en lugar de considerar instancias anidadas, se escogen aleatoriamente 3 grupos para cada uno de los tamaños considerados ( 3, 5, 8 y 10 ) sobre un total de 15 instancias. Se lanzaron 30 ejecuciones sobre cada grupo. Las condiciones de parada se fijaron a cuatro millones de evaluaciones de la función objetivo por instancia del grupo. Los resultados que se han obtenido podemos verlos en las tablas 3 y 4.

| Caso | 3 instancias | 5 instancias | 8 instancias |
|---|---|---|---|
| Secuencial | 23 | 24 | 21 |
| Caso 1 | *29 | *29 | *29 |
| Caso 2 | *29 | 28 | *28 |
| Caso 3 | 28 | *30 | *28 |
| Caso 4 | 25 | 28 | *28 |

Tabla 1. Nº de ejecuciones exitosas[2]

| Caso | 3 instancias | 5 instancias | 8 instancias |
|---|---|---|---|
| Secuencial | 34,0(16,4) | 22,7(12,3) | 25,6(15,2) |
| Caso 1 | 32,2(23,9) | 26,1(20,0) | 24,3(13,5) |
| Caso 2 | 29,7(19,0) | 27,6(17,6) | 23,3(14,0) |
| Caso 3 | *27,0(21,0) | 28,0(22,7) | 27,2(18,1) |
| Caso 4 | 36,6(22,0) | 29,7(15,4) | 25,7(16,4) |

Tabla 2. Media ( desv. típica ) del porcentaje de evaluaciones

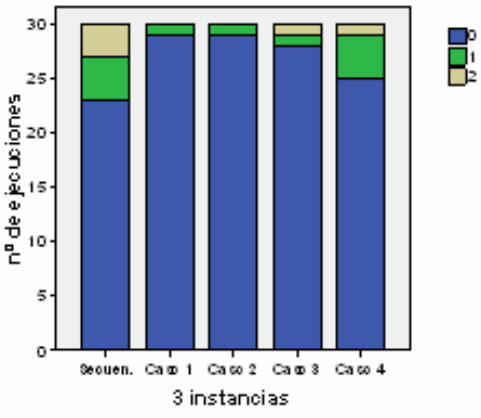


Figura 2.

[2] * Significación < 0,05 con respecto al caso secuencial obtenida mediante el test no paramétrico de Mann-Whitney

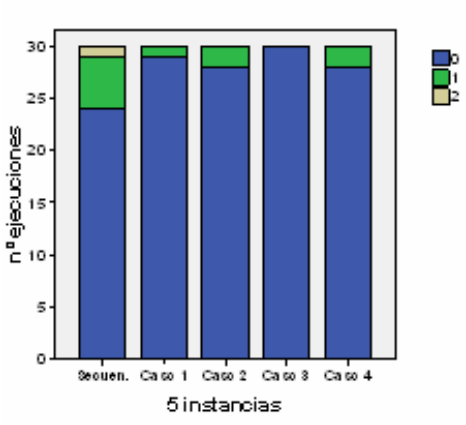


Figura 3.

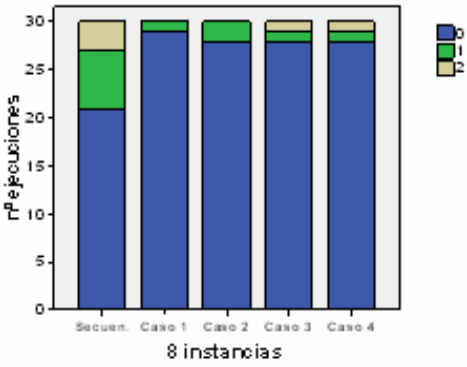


Figura 4.

Se puede comprobar que los resultados siguen la línea de los anteriores. Vemos de nuevo que la búsqueda simultánea es más robusta consumiendo una cantidad similar de recursos. También es interesante resaltar el aumento de la cantidad de ejecuciones exitosas cuando incrementamos el número de instancias. La lectura que podemos hacer de este hecho es que al ampliar la cantidad de instancias, es más probable que haya un mayor número de instancias similares entre ellas y por tanto la adaptación realizada se aproveche en mayor medida, traduciéndose en una búsqueda más eficiente, como muestran los resultados. Siguiendo el esquema del caso anterior, también vamos a estudiar las ejecuciones no exitosas. Los diagramas apilados del número de instancias sin resolver se muestran en las figuras 5, 6, 7 y 8. Podemos ver que la búsqueda simultánea tiene un comportamiento igual o mejor cuando tomamos como referencia el número de instancias no resueltas en la ejecuciones no exitosas.

| Caso | 3 instan. | 5 instan. | 8 instan. | 10 instan. |
|---|---|---|---|---|
| Secuencial | 85 | 81 | 79 | 69 |
| Caso 1 | 87 | 87 | *90 | *85 |
| Caso 2 | 88 | 85 | *90 | *82 |
| Caso 3 | 86 | 87 | *89 | *81 |
| Caso 4 | 88 | *89 | *89 | *85 |

Tabla 3. Nº de ejecuciones exitosas

| Caso | 3 instan. | 5 instan. | 8 instan. | 10 instan. |
|---|---|---|---|---|
| Secuen. | 19,5(9,5) | 27,7(17,9) | 21,6(11,0) | 30,9(14,7) |
| Case 1 | *16,6(12,3) | 26,8(19,7) | *19,3(11,4) | 33,8(19,5) |
| Case 2 | 20,6(12,7) | 25,1(17,1) | 22,1(13,6) | 34,7(20,3) |
| Case 3 | *17,8(13,1) | 27,5(20,0) | 21,9(14,6) | 28,9(14,0) |
| Case 4 | *18,5(12,7) | 32,6(17,9) | 21,8(12,9) | 31,5(16,2) |

Tabla 4. Media ( desv. típica ) del porcentaje de evaluaciones

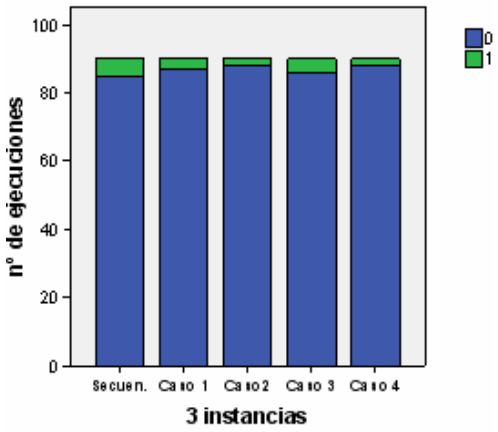


Figura 5.

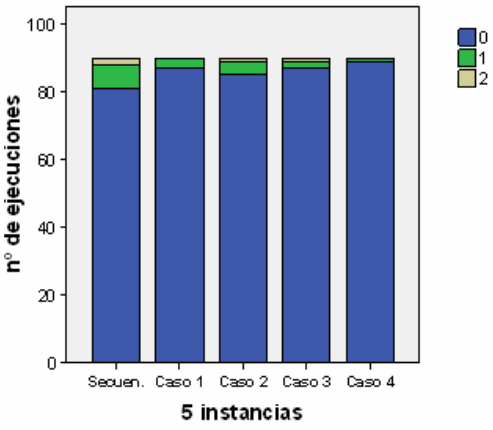


Figura 6.

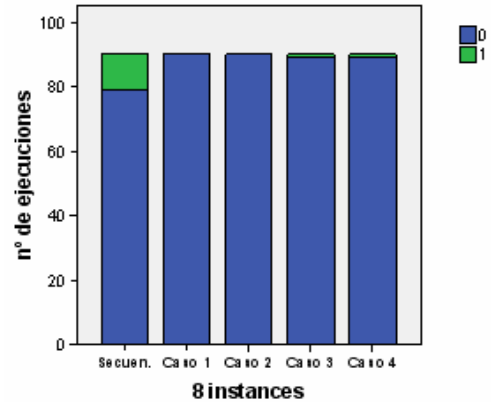


Figura 7.

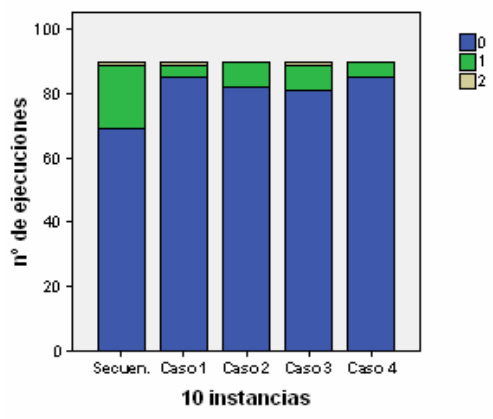


Figura 8.

Podemos concluir señalando que en este artículo hemos analizado el comportamiento de una metaheurística de propósito general que incorpora mecanismos de auto-ajuste . Un aspecto importante que hemos planteado es la posibilidad de resolver un conjunto de instancias simultáneamente, para así aprovechar el esfuerzo computacional que requiere la calibración de los sistemas de auto-ajuste. Los resultados obtenidos muestran como este aprovechamiento se traduce en una búsqueda que consumiendo un número similar de recursos, en términos de evaluaciones de la función objetivo, resulta ser más robusta.

## Referencias


[1] Crawford, J.M. and Auton, L.D. (1996). Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence*, 81(1–2):31–57.

[2] Holger H. Hoos, Stützle, T. (2000): *SATLIB: An Online Resource for Research on SAT*. In: I.P.Gent, H.v.Maaren, T.Walsh, editors, SAT 2000, pp.283-292, IOS Press. SATLIB está disponible online en *www.satlib.org*.

[3] Hoos, H. , Stützle, T.(2000): Local search algorithms for SAT: An empirical evaluation. *J. Automat. Reas.*. 24:421–481.

[4] Mateo, J.L., De la Ossa, L.,(2006): http://www.dsi.uclm.es/simd/software/LiO/

[5] Melián, B., Moreno Pérez, J.A., Moreno Vega, J.M. (2003): Metaheurísticas: Una visión global. *Revista Iberoamericana de Inteligencia Artificial* 19, 2, 7-28

[6] Pelta, D., Blanco, A., Verdegay, J.L. (2002). A fuzzy valuation-based local search framework for combinatorial optimization problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177-193

[7] Selman, B., Kautz, H. A. and Cohen, B. (1994): Noise strategies for improving local search, in *Proceedings of AAAI'94*, MIT Press, 1994, pp. 337–343.

[8] Yokoo, T. Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms (1997): Analyzing Landscapes of CSPs. In *Proceedings of CP'97*, pages 357–370. Springer Verlag.