Un criterio para seleccionar operadores genéticos para resolver CSP

María Cristina Riff Rojas
Universidad Técnica Federico Santa María
Casilla 110-V
Valparaíso
CHILE
E-mail: mcriff@inf.utfsm.cl

Resumen

Nuestro interés es definir algoritmos evolucionistas para la resolución de problemas de satisfacción de restricciones (CSP), los cuales tomen en cuenta tanto las ventajas de los métodos tradicionales de resolución de CSP, asi como las características propias de este tipo de problemas.
En este contexto, se propone en el presente artículo un criterio para poder evaluar la performance de los operadores genéticos dentro de algoritmos evolucionistas que resuelven CSP.

Palabras Claves: *Algoritmos evolucionistas, Satisfacción de restricciones, Operadores genéticos especializados*

Un criterio para seleccionar operadores genéticos para resolver CSP

1 Introducción

Este artículo involucra dos áreas de investigación: Los problemas de satisfacción de restricciones (CSP[1]) en dominios finitos y los métodos evolucionistas. Dentro de estos últimos se consideran en una forma particular los operadores genéticos. Los CSPs son problemas NP-difíciles. Un CSP en dominios finitos consiste en un conjunto de variables, sus dominios y un conjunto de restricciones entre dichas variables. El objetivo es encontrar un valor para cada variable, desde sus respectivos dominios, tal que todas las restricciones sean satisfechas [HE80], [Kum92], [Fre82].

Los métodos evolucionistas están basados en la teoría de la evolución y se clasifican dentro de la categoría de métodos de búsqueda estocástica, para resolver problemas de optimización, [Mic94]. Estos han sido aplicados para resolver Problemas de Optimización con Restricciones (CSOP[2]) [Tsa90], y CSP [BD95], [DBB94], [ERR94], [ERR95], [ERR96], [Rif97c], [Rif97b], [FF95], argumentando que son métodos eficientes para la resolución de problemas con restricciones, de gran tamaño. Los métodos actualmente propuestos en la literatura han concentrado su estudio en la representación genética y en los mecanismos de reproducción (básicamente en los operadores genéticos). Una de las interrogantes actuales más comunes dentro de estos métodos es el cómo poder evaluar la eficiencia de los operadores genéticos.

Por otro lado, la comunidad de investigación sobre restricciones se ha enfocado tradicionalmente a desarrollar técnicas para mejorar la performance de los algoritmos de resolución, usando el conocimiento de las restricciones [Dec90], [HE80], [Fre82], [AB96], por ejemplo podando el árbol de búsqueda. Los algoritmos que realizan una búsqueda sistemática son, en general, evaluados considerando el número de verificaciones de restricciones realizado para encontrar una solución o para decidir que el problema no tiene solución.

Lo que sigue del artículo está organizado de la siguiente manera. Luego de definir algunas nociones sobre los CSP y sobre los algoritmos evolucionistas (EA[3]) en la sección 2, se presenta en la sección 3 dos operadores genéticos que a continuación serán comparados utilizando el modelo propuesto en la sección 4. En la sección 5 se presentan diferentes tests realizados con CSP que tienen solución, los cuales son generados aleatoriamente.

En la sección 6 se presentan algunas conclusiones.

[1] en inglés, Constraint Satisfaction Problems
[2] em inglés: Constraint Satisfaction Optimization Problems
[3] en inglés:Evolutionary Algorithms

2 CSP y EA

En esta sección se mencionan las nociones básicas en CSP, tales como: matriz de restricciones, instanciación e instanciación parcial. A continuación, se presentará la estructura de una algoritmo evolucionista y sus componentes, lo que será utilizado en las siguientes secciones del artículo. Aqui, se restringe la atención a problemas de satisfacción de restricciones binarios, en los cuales las restricciones involucran a lo más dos variables. Es posible convertir un CSP con restricciones n-arias a un equivalente binario [RPD89].

2.1 Conceptos en CSP

Un *Problema de Satisfacción de Restricciones* (CSP) está compuesto de un conjunto de *variables* $V = \{X_1, \ldots, X_n\}$, sus respectivos *dominios* $D_1, \ldots, D_n$ y un conjunto θ que contiene η *restricciones* sobre esas variables. El dominio de una variable es un conjunto de valores con los cuales la variable puede ser instanciada. El tamaño de los dominios son $m_1, \ldots, m_n$, respectivamente, con $\mathbf{m}$ igual al máximo de los m_i. Cada variable X_j es *relevante* (en lo que sigue se denotará *"ser relevante para"* por $\rhd$), para un subconjunto de restricciones $C_{j_1}, \ldots, C_{j_k}$ donde $\{j_1, \ldots, j_k\}$ es alguna subsecuencia de $\{1, 2, \ldots, \eta\}$. Una *restricción binaria* tiene exactamente dos variables relevantes. Un CSP binario tiene asociado un grafo de restricciones, donde los nodos representan las variables y los arcos representan las restricciones. Si las variables relevantes de una restricción dada, tienen valores que no permiten satisfacer la restricción, entonces se dice que ha ocurrido una *inconsistencia* o una violación de la restricción.

Una solución del problema de satisfacción de restricciones consiste de una instanciación de todas las variables la cual no viola ninguna restricción. El algoritmo de resolución más simple es el de la fuerza bruta (generar y probar), el cual intenta simplemente cada posible combinación de valores de las variables.

Definition 2.1 *(Matriz de Restricciones)*
Una Matriz de Restricciones $\mathbf{R}$ *es un arreglo rectangular* $\eta \times n$*, tal que:*

$$\mathbf{R}_{\alpha j} = \mathbf{R}[\alpha, j] = \begin{cases} 1 & \textit{si variable } X_j \rhd C_\alpha \\ 0 & \textit{en otro caso} \end{cases}$$

Remark 2.1 *Con restricciones binarias, hay solo dos valores diferentes de cero en cada fila de* $\mathbf{R}$*.*

Definition 2.2 *(Instantiación)*
Una Instantiación $\mathbf{I}$ *es una asignación desde una n-tupla de las variables* $(X_1, \ldots, X_n) \rightarrow D_1 \times \ldots \times D_n$*, tal que ésta asigna un valor desde su dominio a cada variable en* V*.*

Definition 2.3 *(Instanciación Parcial)*
Dado $V_p \subseteq V$*, una Instanciación Parcial* $\mathbf{I_p}$ *es una asignación desde una j-tupla*

de variables $(X_{p_1}, \ldots, X_{p_j}) \rightarrow D_{p_1} \times \ldots \times D_{p_j}$*, tal que asigna un valor desde su dominio a cada variable en* V_p*.*

Nota: Se hablará de la satisfacción (o insatisfacción) de C_α *bajo* $\mathbf{I_p}$ *solo si todas las variables relevantes de* C_α *están instanciadas*

2.2 Algoritmos Evolucionistas

La estructura de un algoritmo evolucionista se muestra en la Figura 1.
El algoritmo evolucionista utilizado en este artículo genera de forma aleatoria

```
Inicio /* Algoritmo Evolucionista */
 t = 0
 inicializar población P(t)                          (1)
 evaluar los individuos en P(t)                      (4)
 mientras (no sea la condición de término) haga
       inicio
       t=t+1
      Padres = seleccionar-padres-desde P(t-1)       (5)
       Hijos = transformar Padres                    (3)
       P(t) = Hijos
      evaluar P(t)                                   (4)
       fin
 fin-mientras
Fin /* Algoritmo Evolucionista */
```

Figure 1: Estructura de un EA y sus componentes

la población inicial. Para construir un cromosoma los valores de las variables son seleccionados desde sus dominios con una probabilidad uniforme. Se ha utilizado una representación genética no-binaria, porque es la representación más natural para este tipo de problemas. Finalmente el algoritmo de selección es sesgado hacia los mejores individuos.

Remark 2.2 *Una instantiación* $\mathbf{I}$ *corresponde a un cromosoma (individuo) en nuestro algoritmo evolucionista.*

2.2.1 Función de Evaluación

Los métodos evolucionistas han sido generalmente aplicados a la resolución de problemas de optimización. Ellos realizan su búsqueda de una solución guiándose por una función objetivo. En un CSP no se requiere optimizar una función, por lo tanto, para poder utilizar un algoritmo evolucionista se requiere

definir una función objetivo ad-hoc. Una posibilidad consiste en buscar maximizar el número de restricciones satisfechas [Fre95]. En [Rif96] se propuso una función de evaluación definida específicamente para CSP, la cuál será utilizada en el algoritmo evolucionista del presente artículo y que se describe suscintamente a continuación:

Definition 2.4 *(Variables involucradas)*
Para un CSP y una instantiación $\mathbf{I}$ *de sus variables, se define un conjunto de variables involucradas* $\mathbf{E}_{\alpha(\mathbf{I})} \subseteq V$ *para cada restricción* C_α $(\alpha = 1, \ldots, \eta)$*, como sigue:*

- $\mathbf{E}_{\alpha(\mathbf{I})} = \phi$ *ssi* C_α *está satisfecha*
- $X_i \in \mathbf{E}_{\alpha(\mathbf{I})}$ *si* $X_i \rhd C_\alpha$ *y* C_α *no está satisfecha bajo* $\mathbf{I}$
- $X_j \in \mathbf{E}_{\alpha(\mathbf{I})}$ *si* $X_i \rhd C_\alpha$ *y* $\exists C_\beta \neq C_\alpha$ *tal que tanto* X_i *como* $X_j \rhd C_\beta$*, y* C_α *no está satisfecha bajo* $\mathbf{I}$

Esta definición muestra que hay variables que son relevantes para más de una restricción. Cuando se viola C_α las variables involucradas en su insatisfacción son las variables que se encuentran directamente conectadas por C_α, además de las otras variables conectadas a las variables relevantes de C_α. Más precisamente, para una instanciación $\mathbf{I}$, el hecho de cambiar el valor de una variable puede afectar varias restricciones. Es este hecho el que se ha incorporado en la función de evaluación. Antes de definir formalmente la función de evaluación se requiere la siguiente definición:

Definition 2.5 *(Evaluación del error)*
Para un CSP con una matriz de restricciones $\mathbf{R}$ *y una instanciación* $\mathbf{I}$*, se define la Evaluación del error* $\mathbf{e}(C_\alpha, I)$ *para una restricción* C_α *como:*

$$\mathbf{e}(C_\alpha, \mathbf{I}) = \textit{Número de variables en } \mathbf{E}_{\alpha(\mathbf{I})}$$

$$\mathbf{e}(C_\alpha, \mathbf{I}) = \begin{cases} \sum_{w=1}^{n} \mathbf{R}[\alpha, w] \left(\sum_{\beta=1}^{\eta} \mathbf{R}[\beta, w] \right) & \textit{si } C_\alpha \textit{ es insatisfecha} \\ 0 & \textit{en otro caso} \end{cases} \quad (1)$$

Remark 2.3 *Si una restricción binaria* C_α *tiene* X_k *y* X_l *como variables relevantes y la restricción no es satisfecha, entonces:*

$\mathbf{e}(C_\alpha, \mathbf{I})$ *= (Número de variables relevantes de* C_α*) + (Efecto de Propagación* X_k *y* X_l*)*
donde el efecto de propagación X_k *y* X_l *en una red de restricciones binaria, se define como el número de restricciones* C_β*,* $\beta = 1, \ldots, \eta, \beta \neq \alpha$ *que tienen ya sea* X_k *o* X_l *como variables relevantes.*

Definition 2.6 *(Contribución a la evaluación)*
Dado un CSP, se define la contribución de C_α *a la función de evaluación* $\mathbf{c}(C_\alpha)$ *como:*

$$\mathbf{c}(C_\alpha) = \mathbf{e}(C_\alpha, \mathbf{I_j}), \text{ cuando } C_\alpha \text{ es violada bajo } \mathbf{I_j}.$$

Remark 2.4 *El valor de* $\mathbf{c}(C_\alpha)$ *no depende de los valores de la instanciación sino del hecho que la instanciación viola la restricción.*

Finalmente, la función de evaluación es la suma de la *evaluación del error* (equation 1) de todas las restricciones en el CSP, es decir:

Definition 2.7 *(Function Objetivo)*
Para un CSP con una matriz de restricciones $\mathbf{R}$ *y una instanciación* $\mathbf{I}$*, y la evaluación del error* $\mathbf{e}(C_\alpha, \mathbf{I})$ *para cada restricción* $C_\alpha, (\alpha = 1, \ldots, \eta)$*, se define una Función Objetivo* $\mathbf{Z}(\mathbf{I})$ *como:*

$$\mathbf{Z}(\mathbf{I}) = \sum_{\alpha=1}^{\eta} \mathbf{e}(C_\alpha, \mathbf{I}) \tag{2}$$

El objetivo es minimizar la función objetivo $\mathbf{Z}(\mathbf{I})$, la cual será igual a cero cuando todas las restricciones sean satisfechas. La función de evaluación refleja que es más importante satisfacer aquellas restricciones que involucran más variables, es decir, aquellas que están más fuertemente conectadas con otras restricciones.

3 Operadores Genéticos

En esta sección se desea presentar una comparación entre dos algoritmos evolucionistas que difieren sólo en sus operadores genéticos. Se presentarán en forma resumida los operadores genéticos que han sido tomado en cuenta.

3.1 Operador asexuado: (n, p, g)

Es un operador especializado definido por Eiben en [ERR95] de la siguiente manera: El operador selecciona un número de posiciones del padre, y luego selecciona nuevos valores para esas posiciones. Los parámetros a definir del operador asexuado corresponden al número de valores a modificar, el criterio para identificar las posiciones de los valores a ser modificados y el criterio para definir los nuevos valores para el hijo. De esta forma, el operador asexuado se denota por (n,p,g) donde n indica el número de valores modificados, y los valores de p y g son elegidos del conjunto {r,b}, donde r indica una selección aleatoria uniforme y b indica alguna selección sesgada basada en heurística. Los

mejores parámetros encontrados para este operador son (n,p,g) = $(\#, r, b)$ donde # significa que el número de valores a ser modificado se elige aleatoriamente, pero éste es a lo más $\frac{1}{4}$ del total de posiciones.

3.2 Arc-operadores

En [Rif97a] se propuso dos operadores denominados en forma genérica *arc-operadores*, los cuales se basan en la estructura del grafo de restricciones. El nombre de *arc-operadores* viene del hecho de que cada restricción se encuentra representada por un arco en el grafo de restricciones.

3.2.1 Arc-crossover

Este operador realiza un cruzamiento entre dos individuos seleccionados aleatoriamente desde la población y crea a partir de ellos un nuevo individuo (su hijo). El hijo hereda sus valores usando un procedimiento glotón[4], el cual analiza restricción por restricción según un orden pre-establecido. El primer arco que analiza es aquel que es el más dificil de satisfacer o lo que es lo mismo, aquel que se encuentra más fuertemente conectado. En términos de las funciones definidas en la sección previa, la primera restricción a tomar en cuenta es aquella que tiene el mayor valor de la función de evaluación del error, cuando la restricción está siendo violada. Eso es lo que respecta al orden del análisis, en lo que concerne la asignación de valores la tarea es un poco más compleja y requiere de ciertas definiciones previas:

Definition 3.1 *(S_α)*
Dado un CSP y una instanciación parcial $\mathbf{I_p}$. *Para una restricción completamente instanciada* C_α *(i.e. todas sus variables relevantes están instanciadas), se define un conjunto* $\mathbf{S}_\alpha \subseteq \theta$ *por* $C_\beta \in \mathbf{S}_\alpha$ *ssi*

- $\exists X_i : X_i \rhd C_\alpha$ *y* $X_i \rhd C_\beta$ *(C_α, C_β tienen una variable en común)*
- $\forall X_j$ *relevante para* C_β, X_j *está instanciada*

Esta definición muestra que el hecho de cambiar el valor de una variable podría eventualmente afectar otras restricciones. Esta es usada en la medida que el algoritmo va construyendo el hijo, es decir, en la medida que se van instanciando las variables del hijo.

Nota:obviamente $C_\alpha \in \mathbf{S}_\alpha$

Definition 3.2 *(Función de evaluación parcial de cruzamiento)*
Dado un CSP con una instanciación parcial $\mathbf{I_p}$, *y los conjuntos* $\mathbf{S}_\alpha$, *y las funciones* $\mathbf{e}(C_\alpha, \mathbf{I_p})$. *Para toda restricción completamente instanciada* C_α, *se define la* función de evaluación parcial de cruzamiento, $\mathbf{cff}(C_\alpha)$ *para* C_α *como:*

[4] en inglés: greedy procedure

$$\mathbf{cff}(C_\alpha, \mathbf{I_p}) = \sum_{C_\gamma \in \mathbf{S}_\alpha} \mathbf{e}(C_\gamma, \mathbf{I_p}) \tag{3}$$

Remark 3.1 *Se extiende el dominio de las funciones definidas en las secciones precedentes para* $\mathbf{I}$ *a* $\mathbf{I}$ *e* $\mathbf{I_p}$ *calculando esas funciones solo considerando aquellas restricciones involucradas (relacionadas con* C_α*) y cuyas variables están instanciadas en* $\mathbf{I_p}$

Definition 3.3 *(*$\mathbf{M_j}$*)*
Dado un CSP con una instanciación parcial $\mathbf{I_p}$*. Para una variable instanciada* X_j *se define un conjunto de restricciones* $\mathbf{M_j} \subseteq \theta$ *por* $C_\alpha \in \mathbf{M_j}$ *ssi:*

- $X_j \rhd C_\alpha$
- C_α *está completamente instanciada bajo* $\mathbf{I_p}$ *(i.e.* $\forall X_k \rhd C_\alpha : X_k$ *está instanciada)*

Definition 3.4 *(Función de evaluación parcial de mutación)*
Dado un CSP con una instanciación parcial $\mathbf{I_p}$*, los conjuntos* $\mathbf{M_j}$ *para todas las variables instanciadas bajo* $\mathbf{I_p}$*, y las funciones* $\mathbf{e}(C_\alpha, \mathbf{I_p})$*. Para toda restricción completamente instanciada* C_α*, se define la* función de evaluación parcial de mutación*,* **mff** *para* X_j *como:*

$$\mathbf{mff}(X_j, \mathbf{I_p}) = \sum_{C_\gamma \in \mathbf{M_j}} \mathbf{e}(C_\gamma, \mathbf{I_p}) \tag{4}$$

Remark 3.2 *Esta función se calcula solo considerando las restricciones involucradas (relacionadas con* X_j*) y cuyas variables están instanciadas en* $\mathbf{I_p}$

Definition 3.5 *(Número de violaciones)*
Dado un CSP y las funciones de evaluación del error $\mathbf{e}(C_\alpha, \mathbf{I_1})$ *asociadas a cada restricción* C_α*, bajo una instanciación* $\mathbf{I_1}$*, y* $\mathbf{e}(C_\alpha, \mathbf{I_2})$ *para cada restricción* C_α*, bajo una instanciación* $\mathbf{I_2}$*. Se define para cada restricción* C_α *el número de violaciones* $\mathbf{nv}(C_\alpha, \mathbf{I_1}, \mathbf{I_2})$ *como:*

$$\mathbf{nv}(C_\alpha, \mathbf{I_1}, \mathbf{I_2}) = \begin{cases} 0 & \mathbf{e}(C_\alpha, \mathbf{I_1}) = \mathbf{e}(C_\alpha, \mathbf{I_2}) = 0 \\ 1 & \textit{ya sea } \mathbf{e}(C_\alpha, \mathbf{I_1}) \neq 0 \textit{ o } \mathbf{e}(C_\alpha, \mathbf{I_2}) \neq 0\textit{, pero no ambas} \\ 2 & \mathbf{e}(C_\alpha, \mathbf{I_1}) \neq 0\textit{, y } \mathbf{e}(C_\alpha, \mathbf{I_2}) \neq 0 \end{cases}$$

La figura 2. muestra el algoritmo de *arc-crossover*.
La idea es la siguiente: Se seleccionan dos individuos de la población y se usarán sus valores para construir un hijo, el cuál "se espera", satisfacerá más restricciones que sus padres. El algoritmo comienza con la restricción más conectada dentro del grafo y determina si ésta es satisfecha o violada por los padres. En

el caso que esté satisfecha por ambos padres, el hijo tomará los valores para las dos variables relevantes de esa restricción, del padre que tiene una mejor evaluación entre los dos. Si sólo un padre satisface esa restricción, entonces el hijo hereda los valores de las dos variables a partir de ese padre. En el caso en que ambos padres violan la restricción, el algoritmo toma los valores de los padres de cada variable que pertenece a esa restricción y los cruza (teniendo por lo tanto dos posibilidades). Dentro de esas dos posibilidades de elección para los valores del hijo, se selecciona aquella que le dará una mejor evaluación al heredero (tomando en cuenta las variables que ya tenga instanciadas), es decir se hace una revisión parcial en el hijo antes de instanciar los valores de las variables. Una vez que casi todas las restricciones han sido analizadas, van a quedar algunas restricciones que considerar. Es probable que algunas restricciones, sin haber sido aún analizadas, tengan ya los valores de sus variables relevantes instanciadas en el hijo (debido a la conectividad con las restricciones analizadas previamente). En el caso en que a alguna restricción le falte sólo una variable por instanciar en el momento que le corresponda su análisis, la selección del valor de esa variable se realiza tomando en cuenta la evaluación que tendría el hijo con el valor proveniente de cada padre, eligiendo aquel que será mejor para el heredero, considerando las variables ya instanciadas.

4 Comparando los Operadores Genéticos

4.1 Número de verificaciones de restricciones

En esta sección se estima el número de verificaciones de restricciones realizadas por un algoritmo que utiliza *arc-mutation* y *arc-crossover* y se compara con un algoritmo que utiliza $(\#, r, b)$ y mutación. Dado un individuo *arc-mutation* selecciona aleatoriamente la variable a modificar. El valor de la variable es elegido, evaluando para cada valor desde su dominio la función de evaluación parcial de mutación dada por la definición 3.4. El valor seleccionado será aquel con el menor valor de **mff**. A continuación presentaremos el modelo utilizado para realizar la comparación de los dos algoritmos.

4.1.1 El modelo

Parámetros del modelo:

- p_c probabilidad de cruzamiento
- p_m probabilidad de mutación
- t_p tamaño de la población
- n número de variables
- m tamaño del dominio

Algoritmo `Arc-Crossover` $(Padre_1, Padre_2)$
Inicio
Para `cada` C_α `siguiendo el orden de la Evaluación del error`
`Analizar` C_α (x_i, x_j) `del` $Padre_1$ `y del` $Padre_2$
si $((X_i \dashv \mathbf{I_p})^a$ `y` $(X_j \dashv \mathbf{I_p}))$ **entonces**
  si $(\mathbf{nv}(C_\alpha, \mathbf{Padre_1}, \mathbf{Padre_2}) = 0)$ **entonces**
    si $(\mathbf{Z}(Padre_2) > \mathbf{Z}(Padre_1))$ **entonces**
      $\mathbf{I_p}(X_i, X_j) = (x_{i_1}, x_{j_1})$
    sino
      si $(\mathbf{Z}(Padre_1) > \mathbf{Z}(Padre_2))$ **entonces**
        $\mathbf{I_p}(X_i, X_j) = (x_{i_2}, x_{j_2})$
      sino
        $\mathbf{I_p}(X_i, X_j) = random((x_{i_1}, x_{j_1}), (x_{i_2}, x_{j_2}))$
  sino
    si $(\mathbf{nv}(C_\alpha, \mathbf{Padre_1}, \mathbf{Padre_2}) = 1)$ **entonces**
      si $(C_\alpha \models Padre_1)^b$ **entonces**
        $\mathbf{I_p}(X_i, X_j) = (x_{i_1}, x_{j_1})$
      sino $\mathbf{I_p}(X_i, X_j) = (x_{i_2}, x_{j_2})$
    sino
      $\mathbf{I_p}(X_i, X_j) = argmin_{s_1 \in S_1}(\mathbf{cff}(C_\alpha, (\mathbf{I_p} \cup (x_{i_1}, x_{j_2}))),$
        $\mathbf{cff}(C_\alpha, (\mathbf{I_p} \cup (x_{i_2}, x_{j_1}))))^c$
sino
  si $((X_i \dashv \mathbf{I_p})$ `o` $(X_j \dashv \mathbf{I_p}))$ **entonces**
    si $(X_i \dashv \mathbf{I_p})$ **entonces** `k=i` **sino** `k=j`
   $\mathbf{I_p}(X_k) = argmin_{s_2 \in S_2}(\mathbf{mff}(X_k, (\mathbf{I_p} \cup x_{k_1})), \mathbf{mff}(X_k, (\mathbf{I_p} \cup x_{k_2})))^d$
Fin

---

[a] X_i no-está-instanciada en $\mathbf{I_p}$
[b] $\models$: satisfecha por
[c] $argmin_{s \in S}\{a_s\}$ da s^* tal que $a_{s^*} \leq a_s, \forall s \in S$.
$S_1 = \{(x_{i_1}, x_{j_2}), (x_{i_2}, x_{j_1})\}$
[d] $S_2 = \{x_{k_1}, x_{k_2}\}$

Figure 2: Estructura del algoritmo Arc-crossover

- p_1 probabilidad de conectividad, es decir, la probabilidad de que exista una restricción entre dos variables.

Consecuencias del modelo:

- Número promedio de restricciones = $\frac{n(n-1)p_1}{2}$
- Conectividad promedio de cada variable = $p_1(n-1)$

arc-crossover: Se va a analizar el peor caso para *arc-crossover*, es decir, cuando la restricción C_α es violada por ambos padres y las dos variables involucradas en C_α no han sido aún instanciadas en el hijo. En ese caso *arc-crossover* debe elegir entre las dos combinaciones de valores desde sus padres, con dos evaluaciones de **cff**. Cada restricción es verificada sólo una vez por combinación de valores. Por lo tanto se tiene la siguiente desigualdad:

- $2\eta \leq$ número de verificaciones de restricciones de arc-crossover$\leq 4\eta$

En consecuencia el número mínimo de verificaciones de restricciones para arc-crossover es 2η, esto es cuando al menos uno de los padres satisface todas las restricciones de la red.

arc-mutation: Evalúa con **mff** todos los otros valores del dominio de la variable (salvo el actual), por lo tanto

- Número de verificaciones de restricciones de arc-mutation = $2\eta(m-1)p_m$

En consecuencia, el número de verificaciones de restricciones para el algoritmo completo que utiliza los *arc-operadores* (N_{valgo}) es:

- $N_{valgo} \leq \left(4\eta\frac{p_c}{2-p_c} + 2\eta(m-1)p_m\right) t_p$

$(\#, \mathbf{r}, \mathbf{b})$ es el operador asexuado. Por lo tanto, el número de verificaciones de restricciones para $(\#, r, b)$ es igual a $\frac{\eta m}{2}$. Para el algoritmo completo que utiliza $(\#, r, b)$ y mutación el número de verificaciones de restricciones es igual a:

- $N'valgo = \frac{t_p p_c \eta m}{2}$

ya que, el operador de mutación standard no verifica ninguna restricción.

Obviamente, el número de verificaciones de restricciones para los *arc-operadores* es mucho mayor que para el otro algoritmo. Sin embargo, la performance de un algoritmo está relacionada con:

- Porcentaje de casos que el algoritmo encuentra una solución
- Número de generaciones para encontrar una solución

5 Tests

Se ejecutaron los dos algoritmos para resolver el clásico problema de satisfacción de restricciones de colorear un grafo con tres colores (3-color). El problema de 3-color consiste en asignar un color a cada nodo, de tal forma que los nodos adyacentes tengan un color diferente. Para ello, se generaron aleatoriamente 100 instancias (que tienen solución) por conectividad, considerando un porcentaje de conectividad entre [10..90]. Se estableció un máximo número de iteraciones de 500 para el algoritmo con los arc-operadores y un máximo de 1500 para el otro algoritmo. La tabla muestra los tiempos de CPU [seg] promedios para resolver 100 grafos y el número promedio de verificaciones de restricciones.

Restricciones	N'_{valgo}	tiempo-CPU'	N_{valgo}	tiempo-CPU
30	11141	67	7609	18
45	185671	1144	44967	110
60	843709	4891	422426	1053
90	1401437	6372	465359	917
120	1311153	5908	250505	632
150	1060255	5938	219715	452
180	1070010	5961	219847	580
210	1022748	5693	201619	419
240	936505	5296	220700	419
270	1153107	5087	252820	673

La performance del algoritmo que utiliza los *arc-operadores* es mucho mejor que la del algoritmo que utiliza $(\#, r, b)$ y mutación, debido a que el primero converge más rapidamente hacia una solución y a que ha sido capaz de encontrar la solución para una mayor cantidad de problemas.

6 Conclusión

Minimizar el número de verificaciones de restricciones es el principal objetivo de los algoritmos que resuelven los CSP de una manera sistemática. Sin embargo, cuando se trata de métodos que realizan una búsqueda estocástica la evaluación de un algoritmo con respecto a otro no es una tarea fácil. Esto se debe a que se trata de algoritmos evolucionistas, donde no se conoce, a priori, el número de iteraciones que realizará el algoritmo, por lo tanto se desconoce cuántas veces será aplicado el (o los) operador(es) genético(s).

En este artículo se ha propuesto, para comparar la eficiencia de los operadores genéticos en la resolución de CSP, un modelo para estimar el número de verificaciones de restricciones realizadas por los operadores. Se propone además, evaluar la performance del algoritmo, en términos del tiempo de CPU y del número real de verificaciones de restricciones como un todo. Esto permite dar una visión de la eficiencia del algoritmo en la resolución de CSP.

Agradecimientos

Se agradece en forma especial al Sr Bertrand Neveu por sus importantes comentarios técnicos. Se agradece además al Prof Dr-Ing. Hans-Paul Schwefel por las facilidades dadas para redactar este artículo durante la visita de investigación en su laboratorio.

References

[AB96] M.S. Affane and H. Bennaceur. A labelling arc consistency method for functional constraints. In Freuder [Fre96], pages 16–30.

[BD95] J. Bowen and G. Dozier. Solving constraint satisfaction problems using a genetic/systematic search hybrid that realizes when to quit. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 122–129, 1995.

[DBB94] G. Dozier, J. Bowen, and D. Bahler. Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm. In *First IEEE International Conference on Evolutionary Computation* [pro94], pages 306–311.

[Dec90] R. Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. In *Artificial Intelligence*, volume 41, pages 273–312, 1990.

[ERR94] A.E. Eiben, P-E Raué, and Zs Ruttkay. Solving constraint satisfaction problems using genetic algorithms. In *First IEEE International Conference on Evolutionary Computation* [pro94], pages 542–547.

[ERR95] A.E. Eiben, P-E Raué, and Zs. Ruttkay. Ga-easy and ga-hard constraint satisfaction problems. In Meyer [Mey95], pages 267–283.

[ERR96] A.E. Eiben, P-E Raué, and Zs. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *Third IEEE International Conference on Evolutionary Computation*, pages 258–261, 1996.

[FF95] C. Fleurent and J. Ferland. Genetic algorithms and hybrids for graph coloring. In *Annals of Operations Research*, 1995.

[Fre82] E. Freuder. A sufficent condition of bactrack-free search. In *Journal of the ACM*, volume 29, pages 24–32, 1982.

[Fre95] E. Freuder. The many paths to satisfaction. In Meyer [Mey95], pages 103–119.

[Fre96] Eugene Freuder, editor. *Constraint Processing (CP96)*, LNCS XXX. Springer-Verlag, 1996.

[HE80] Haralick and Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *Artificial Intelligence*, volume 14, pages 263–313, 1980.

[Kum92] V. Kumar. Algorithms for constraint satisfaction problems:a survey. In *Artificial Intelligence Magazine*, volume 13, pages 32–44, 1992.

[Mey95] Manfred Meyer, editor. *Constraint Processing (CP95)*, LNCS 923. Springer-Verlag, 1995.

[Mic94] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Artificial Intelligence. Springer-Verlag, 1994.

[pro94] *First IEEE International Conference on Evolutionary Computation*, 1994.

[pro97] *Fourth IEEE International Conference on Evolutionary Computation*, 1997.

[Rif96] M.-C. Riff. From quasi-solutions to solution: An evolutionary algorithm to solve csp. In Freuder [Fre96], pages 367–381.

[Rif97a] M.-C. Riff. Evolutionary search guided by the constraint network to solve csp. In *Fourth IEEE International Conference on Evolutionary Computation* [pro97], pages 337–342.

[Rif97b] M.-C. Riff. Self-adapting crossover operator for csp. In *3eme Conference Nationale pour la Resolution Pratique de Problemes NP-Complets*, pages 41–48, 1997.

[Rif97c] M.-C. Riff. Using the knowledge of the constraints network to design an evolutionary algorithm that solves csp. In *Fourth IEEE International Conference on Evolutionary Computation* [pro97], pages 279–284.

[RPD89] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problem. Act-ai-222-89, MCC Corporation, Austin, Texas, 1989.

[Tsa90] E. Tsang. Applying genetic algorithms to constraint satisfaction optimization problems. In *Nineth European Conference on Artificial Intelligence*, pages 649–654, 1990.