Cómputo Evolutivo

Reporte Exposición : Estrategia Evolutiva ($\mu + \lambda$), Comparación de operadores de recombinación

Diego Dozal Magnani : 316032708

June 1, 2023

Estrategias Evolutivas

Contexto histórico

Las Estrategias Evolutivas (ES) fueron propuestas por Rechenberg y Schwefel en el 1960 y la estrategia más temprana fue *ES-(1+1)*, donde sólo había un padre y una generación de un solo individuo (hijo mutado), por lo anterior también se conoce como clonación. Después, uno de los avances má importantes en las ES fue que Rechenberg propuso la conocida regla de $\frac{1}{5}$. Y en 1970 el concepto de estrategia evolutiva multi-miembro. Con la convención de que μ individuos conforman la población de posibles padres, y λ individuos conforman la descendencia (offspring) en cada ciclo.[3]

Descripción de las Estrategias Evolutivas

El objetivo de una ES es optimizar una función objetivo F respecto a un conjunto de parámetros objetivos $y = (y_1, ..., y_n)$. Con y un vector de valores que pertenece a R^n, Z^n , etc. Buscamos que $F(y) \rightarrow opt$.
Las ES operan sobre poblaciones A de invididuos α, donde cada individuo se compone de

$$\alpha_k =< y_k, s_k, F(y_k) >$$

- y_k : Es el vector de valores sobre un conjunto específico.
- s_k : `parámetros endógenos` (por ejemplo σ).
- $F(y_k)$: Función objetivo de y_k

Los parámetros endógenos (s_k) son usados para controlar ciertas propiedades estadísticas de los operadores genéticos, específicamente aquellos relacionados con la mutación. Adicionalmente, estos pueden evolucionar durante el proceso y son necesarios para la auto-adaptación.
Parámetros exógenos :

- μ : Tamaño de población (de la cual se seleccionan los padres)
- λ : Tamaño de la población de los hijos (offspring).
- ρ : Número de mezcla, el cual se refiere al número de padres involucrados en la procreación de un individuo.

Notación $(\mu/\rho,^+ \lambda)$-ES

1. μ :Número de individuos en cada población
2. λ : Número de individuos de la nueva población.

Cuando ρ¿1 , las estrategias de evolución incluyen estrategias de recombinación. Adicionalmente ρ determina el tamaño de los matrimonios , si $\rho = \lambda$ entonces todos los padres forman parte del matrimonio (también conocida como recombinación global).Finalmente el $+$ y **,** se refiere al tipo de operador de selección.

Operadores a Implementar

Operador de Selección $(\mu+\lambda)$

`Implementación : src/es_mu_plus_lambda.py -> class ESMuPlusLambda.mu_plus_lambda_selection`

Todo algoritmo evolutivo necesita un operador de selección que guíe la búsqueda a regiones más prometedoras del espacio de búsqueda. En contraste a (μ, λ)-ES, $(\mu + \lambda)$-ES considera a los mejores individuos entre los μ individuos de la población actual y a los nuevos λ individuos de la descendencia. Debido a esto, también es llamado *elitismo*. La selección $(+)$ garantiza la supervivencia del mejor individuo encontrado hasta la última iteración.
Lllamaremos *selection pool* $= \mu + \lambda$ al conjunto de individuos candidatos para ser seleccionados.
Una forma de seleccionar a los mejores es ordenar a los individuos que pertenecen a nuestro selection pool respecto a su función objetivo y seleccionar a los primeros μ mejores individuos.[2]

Operador de Mutación

`Implementación : src/es_mu_plus_lambda.py -> class ESMuPlusLambda.mutate`

Por cada individuo $\alpha_k =< y_k, s_k, F(y_k) >$ que forma parte de los λ descendientes generados. Consideramos $s_k = \sigma$ y $y_k \in R^n$.

1. Generamos a r un vector aleatorio de la distribución normal $N(0, \sigma^2)$.
2. Hacemos a $y_k^* = y_k + r_k$
3. Devolvemos al individuo mutado $\alpha_k^* =< y_k^*, s_k, F(y_k) >$

Auto-Adaptación del parámetro endógeno σ

`Implementación : src/es_mu_plus_lambda.py -> class ESMuPlusLambda.execute [194:202], class ESIndividual.i`

Basándonos en la regla de $\frac{1}{5}$ de Rechenberg, en donde el 20% de las mutaciones debería resultar en una mejora de la función de aptitud. Podemos evolucionar la fuerza de mutación basándonos en la tasa de éxito de mutación cada cierta cantidad de iteraciones.

1. Si la tasa de éxito es menor que $\frac{1}{5}$, entonces incrementamos $\sigma = c^2 \cdot \sigma$
2. Si la tasa de éxito es mayor que $\frac{1}{5}$, entonces disminuimos $\sigma = \frac{\sigma}{c}$

Con $c = 0.817$ el factor de cambio.

Operadores de Recombinación

A diferencia de los *GA*, la aplicación del operador de recombinación en las ES a una familia de padres de tamño ρ produce un sólo hijo. Por lo que se tienen que realizar λ iteraciones en orden de generar a la población de hijos. Cuando $\rho > 2$ entonces hay multi recombinación. Hay dos clases estándar de recombinación : *Recombinación Discreta* algunas veces llamada *Recombinación Dominante* y la *Recombinación Intermedia*.

Sea A_ρ el conjunto de padres seleccionados para generar a un individuo $\alpha^{**} =< y^{**} = [], \sigma^{**}, F(y^{**}) >$ y $\alpha_j \in A_\rho$.

- **Recombinación Discreta o Dominante**

  `Implementación : src/es_mu_plus_lambda.py -> class ESMuPlusLambda.discrete_recombination`

  1. De $i = 0$ hasta $|\alpha_j.y_i|$ generamos un número aleatorio $r_i$entre $(0, |A_{\rho|})$. Esto será para seleccionar el padre del cual obtendremos la información del vector de valores:
     (a) Obtenemos $\alpha_r \in A_\rho$.
     (b) Hacemos $\alpha^{**}.y^{**}[i] = \alpha_r.y_r[i]$. Es decir asignamos el elemento del vector de valores del padre seleccionado al nuevo hijo.
  2. Seleccionamos estocásticamente uno de los parámetros endógenos σ de los individuos de A_ρ, sea σ_i. Hacemos $\sigma^{**} = \sigma_i$
  3. Regresamos α^{**}

- **Recombinación Intermedia**

  `Implementación : src/es_mu_plus_lambda.py -> class ESMuPlusLambda.intermediate_recombination`

  1. De $i = 0$ hasta $|\alpha_j.y_i|$ obtenemos el valor promedio por cada $\alpha_j \in A_\rho$ en la posición i, sea este y_{avg}. Hacemos $y^{**}[i] = y_{avg}$.
  2. Sea σ_{avg} el promedio de los parámetros endógenos para cada invidiuo en A_ρ. Hacemos $\sigma^{**} = \sigma_{avg}$.
  3. Regresamos α^{**}

A continuación se presenta un diagrama en donde se resume el proceso de las recombinaciones.

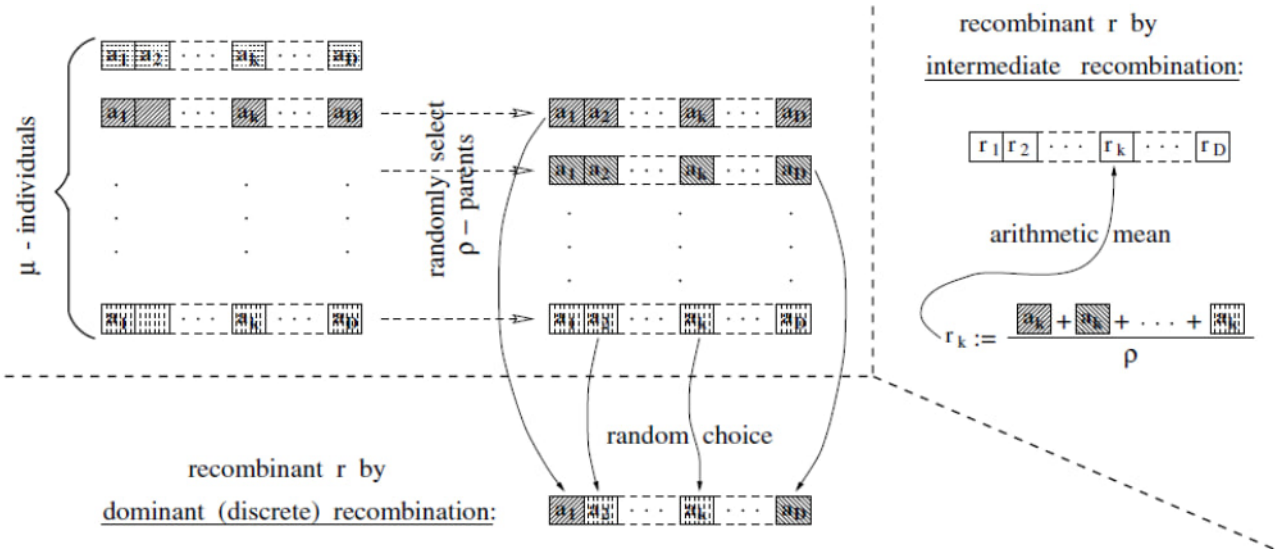

Figure 1: Comparativa de operadores de recombinación[3]

Problema

Objetivo de implementación

La intención de este reporte es mostrar los resultados obtenidos al implementar la estrategia $(\mu+\lambda)-$ES para optimizar la función *Ackley* por medio de comparar dos operadores distintos de recombinación, en particular **recombinación discreta** y **recombinación intermedia**. Los parámetros de entrada son los mismos para ambos operadores y adicionalmente, ambos algoritmos usan **recombinación** con el parámetro σ (la cual fue viste en clase).
Los algoritmos se implementan en python.

Representación de soluciones

Como se mencionó en clase y en anteriores secciones, las soluciones son representan por medio de vectores reales de distintas dimensiones, para las pruebas realizadas, se utilizaron dimensiones 2, 5, 10 y 20.

Función Evaluación

Buscamos minimizar la función `Ackley`, esto es acercarse al vector 0.

$$f(x) = 20 + e - 20e\left(-0.2\sqrt{\frac{1}{2}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{2}\sum_{i=1}^{n}(cos2\pi x_i)\right)$$

Función objetivo

La función objetivo es la misma función Ackley, buscamos que la evaluación del vector sea cercana al 0. El rango considerado para el dominio de la función es

Detalles de implementación

Pseudocodigo

Algorithm 1: $(\mu + \lambda)$-ES

Data: $F : R^n \rightarrow R$,$[a \in R, b \in R]$, μ, λ, ρ, m
Result: $y \in R^n$

*[r]m es el umbral de iteraciones para revisar la tasa de éxito `iters` $\leftarrow 0$ *[r]Contador de las iteraciones actuales $s \leftarrow 0$ *[r]Contador de éxitos $c \leftarrow 0.817$ *[r]Factor de cambio $\beta_0 \leftarrow []$ *[r]Población inicial de los μ individuos **for** $i \leftarrow 1$ *to* μ **do**
    *[r]Los individuos son de la forma $\alpha =$ ¡$y \in R^n$, $F(y)$, σ¿ $\beta_0.append($`randomIndividualGenerator(`n`, `a`, `b`, `σ`)`$)$
while *Halt Condition* **do**
    $\beta_\rho \leftarrow []$ *[r]Individuos seleccionados para ser padres **for** $j \leftarrow 1$ *to* ρ **do**
        $\beta_\rho.append($`randomSelection(`β_0`)`$)$ *[r]Selecciona un individuo aleatoreamente de β_0
    *[r]Procedemos a la recombianción $\beta_\lambda \leftarrow []$ *[r]Individuos generados a partir de la recombinación **for** $j \leftarrow 1$ *to* λ **do**
        $\alpha =$`discreteRecombination(`β_ρ`)`;
        OR;
        $\alpha =$`intermediateRecombination(`β_ρ`)`;
        $\beta_\lambda.append(\alpha)$
    *[r]Procedemos a la mutación
    for $j \leftarrow 1$ *to* λ **do**
        $\alpha^{**} =$`mutate(`$\beta_\lambda[j]$`)`;
        if $F(\alpha^{**})$ *isbetterthan* $F(\beta_\lambda[j])$ **then**
            $\beta_\lambda[j] \leftarrow \alpha^{**}$;
            $s++$ *[r]Incrementamos el contador de éxitos *[r]Procedemos a la autoadaptación **if** *iters* $== m$ **then**
                *[r]Si alcanzamos un umbral de iteraciones **if** $\frac{s}{m} < \frac{1}{5}$ **then**
                    $\beta_\lambda[j].\sigma \leftarrow c^2 \cdot \sigma$
                else
                    $\beta_\lambda[j].\sigma \leftarrow \frac{\sigma}{c}$
                `iters`$\leftarrow 0$;
                $s \leftarrow 0$
    *[r]Procedemos a realizar la selección $(\mu + \lambda)$ $\beta_{\mu+\lambda} \leftarrow \beta_0 \cup \beta_\lambda$;
    `iters ++ ;`
    $\beta_0 \leftarrow$ `nueva poblacion`

Algorithm 2: $(\mu + \lambda)$-ES

Data: $F : R^n \rightarrow R$,$[a \in R, b \in R]$, μ, λ, ρ, m
Result: $y \in R^n$

while *Halt Condition* **do**
  *[r]Procedemos a la mutación **for** $j \leftarrow 1$ *to* λ **do**
    α^{**} =`mutate(`$\beta_\lambda[j]$`)`;
    if $F(\alpha^{**})$ *isbetterthan* $F(\beta_\lambda[j])$ **then**
      $\beta_\lambda[j] \leftarrow \alpha^{**}$;
      $s++$ *[r]Incrementamos el contador de éxitos *[r]Procedemos a la autoadaptación **if** *iters* $== m$ **then**
        *[r]Si alcanzamos un umbral de iteraciones **if** $\frac{s}{m} < \frac{1}{5}$ **then**
          $\beta_\lambda[j].\sigma \leftarrow c^2 \cdot \sigma$
        else
          $\beta_\lambda[j].\sigma \leftarrow \frac{\sigma}{c}$
        `iters`$\leftarrow 0$;
        $s \leftarrow 0$
  *[r]Procedemos a realizar la selección $(\mu + \lambda)$ $\beta_{\mu+\lambda} \leftarrow \beta_0 \cup \beta_\lambda$;
  `sortByFitness(`$\beta_{\mu+\lambda}$`)` *[r]Ordenamos el total de inviduos de $\beta_{\mu+\lambda}$ respecto a F $\beta_0 \leftarrow []$ *[r]Reiniciamos la población **for** $i \leftarrow 1$ *to* μ **do**
    $\beta_0.append(\beta_{\mu+\lambda}[i])$
  `iters ++ ;`
returns `theBestOf(`β_0`)`

Experimentación

Tabla de parámetros

μ	λ	ρ	m : `Self-Adaptation threshold`	`Repetitions`	`Iterations`
20	100	10	10	10	100

Resultados

La siguiente tabla fue generada con los resultados onbtenidos al realizar cada ejecución por 10 repeticiones.

Recombination	Dimention	Fitness(Best)	Fitness(Worst)	AVG Time (s)
Discrete	2	0.00277909	10.12035279	1.16361467
Intermediate	2	0.1731236	14.9228724	1.17941527
Discrete	5	0.066216269	7.4107573619	1.3925580
Intermediate	5	1.40504689	13.5132380	1.373049044
Discrete	10	0.805549307	9.105042154	2.54239213
Intermediate	10	3.46406131	12.01281135	4.49303894
Discrete	20	3.464061	12.0128113	4.49303894
Intermediate	20	3.552713678e-15	3.552713678e-15	2.1663813

Table 1: Ejecuciones con condición de paro **límite de tiempo.**

Gráficas

Ejecuciones Individuales

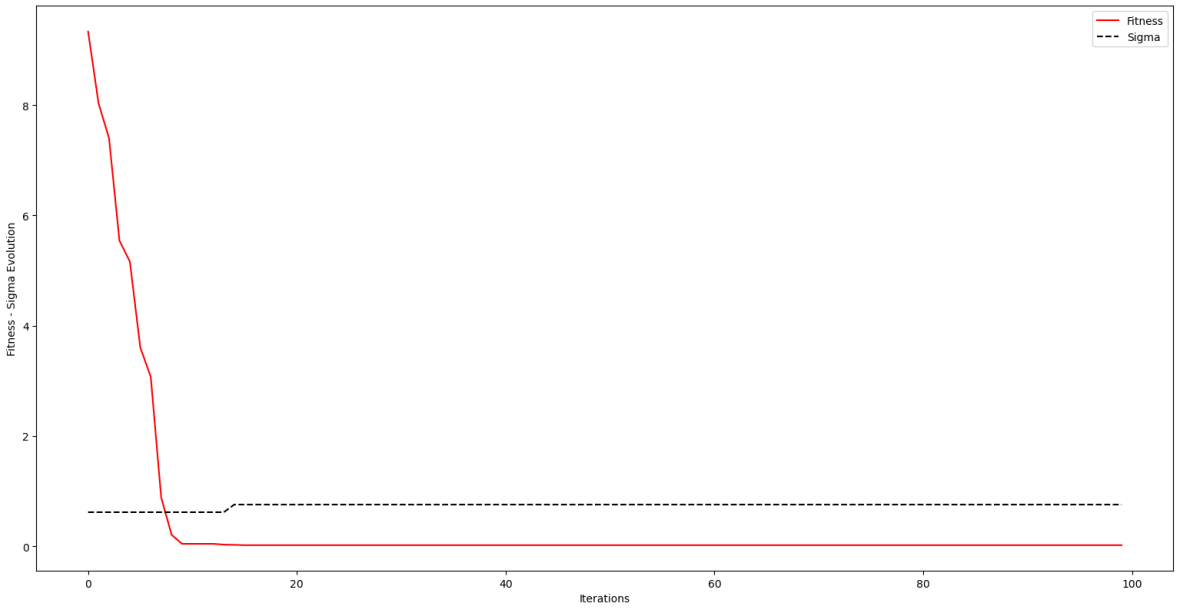

Figure 2: Ejecución individual discreta dimensión 2

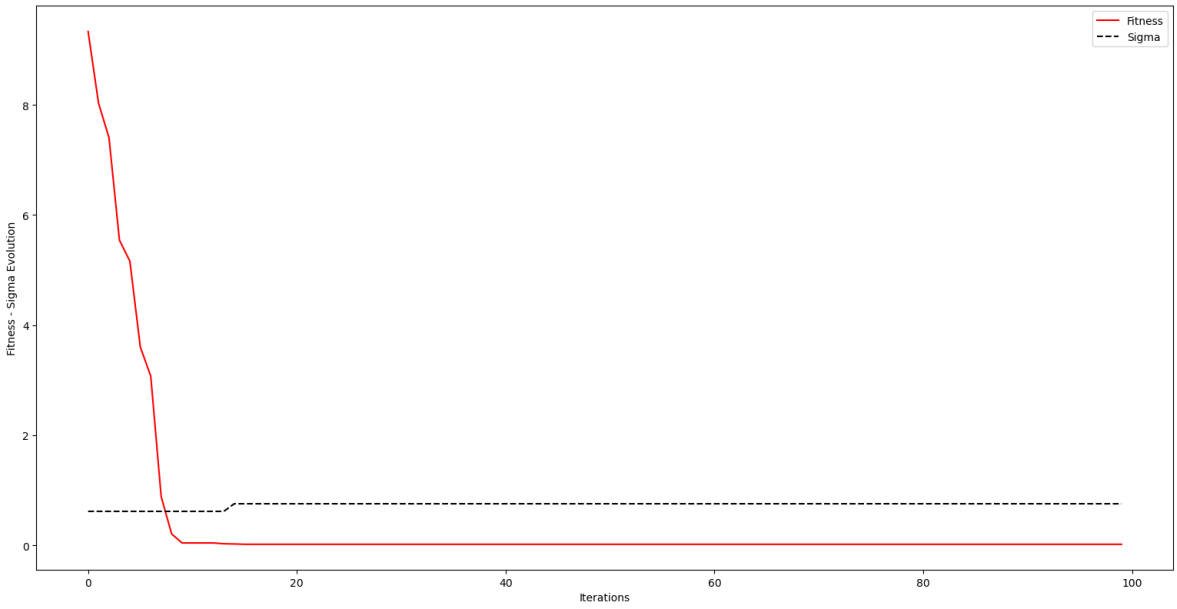

Figure 3: Ejecución individual intermedia dimensión 2

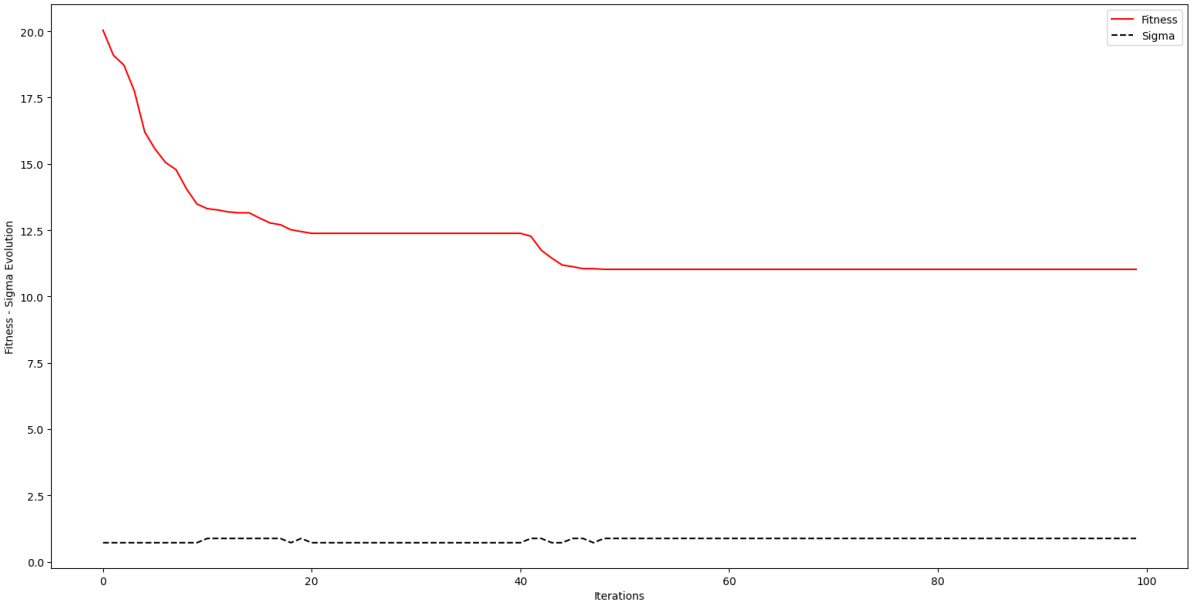

Figure 4: Ejecución individual intermedia dimensión 20

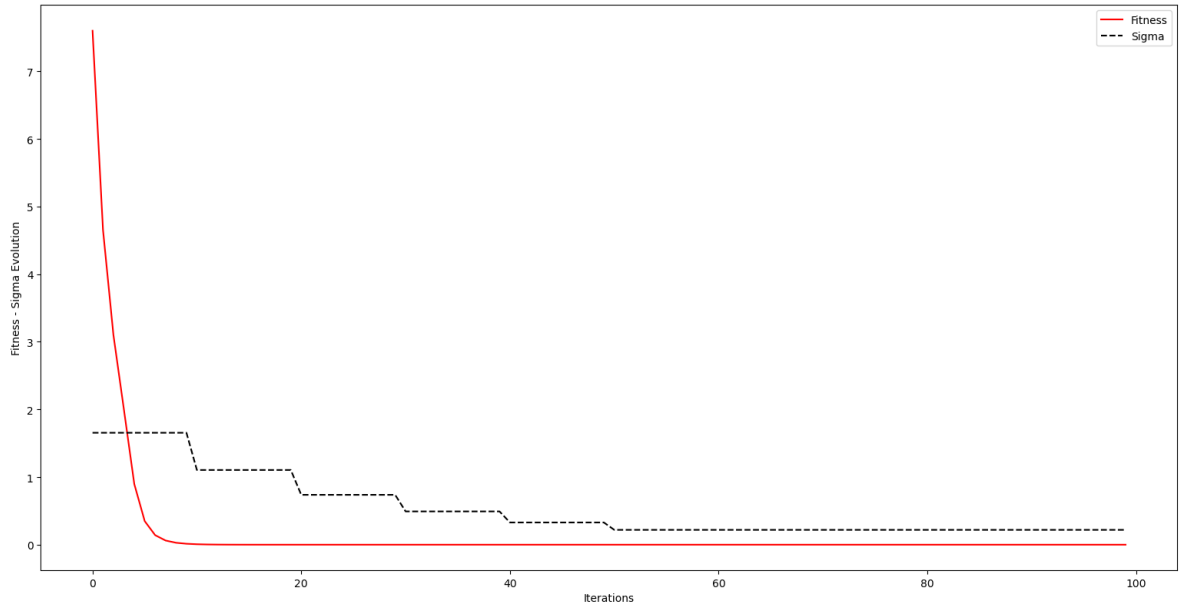

Figure 5: Ejecución individual intermedia dimensión 20

En las anteriores ejecuciones individuales, podemos observar que al contrastar los operadores de recombinación en dimensión 2, parece ser que la recombinación discreta tiene mejor desempeño que la intermedia. Por otro lado, en dimensión 20 la recombinación intermedia encuentra muy buenos resultados mientras que la discreta no alcanza valores cercanos.

Evolución Promedio

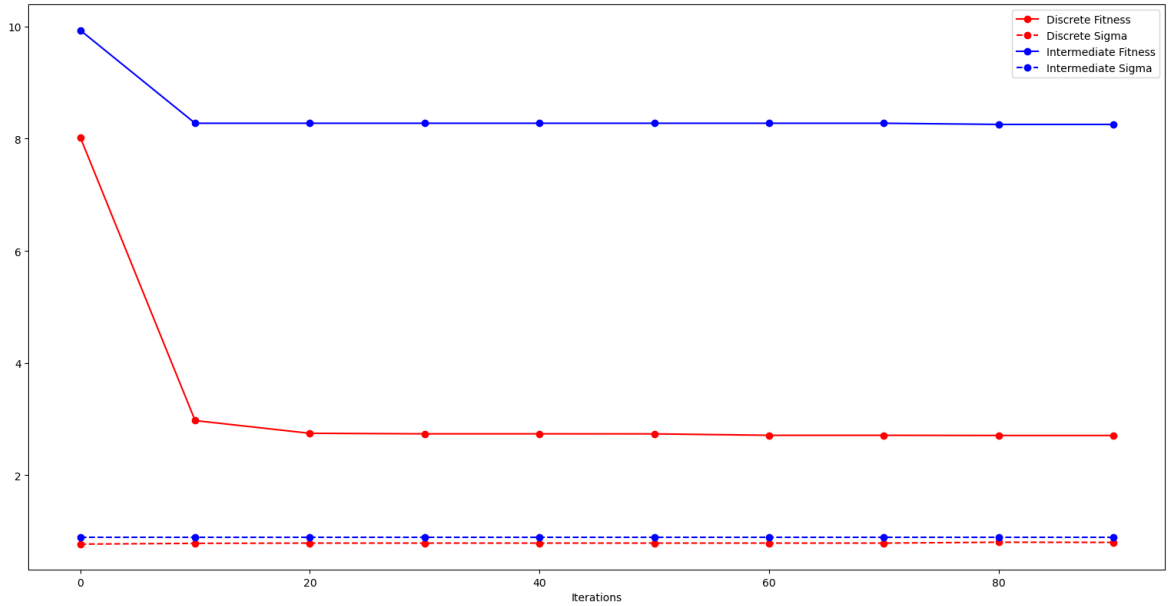

Figure 6: Evolución promedio dimensión 2

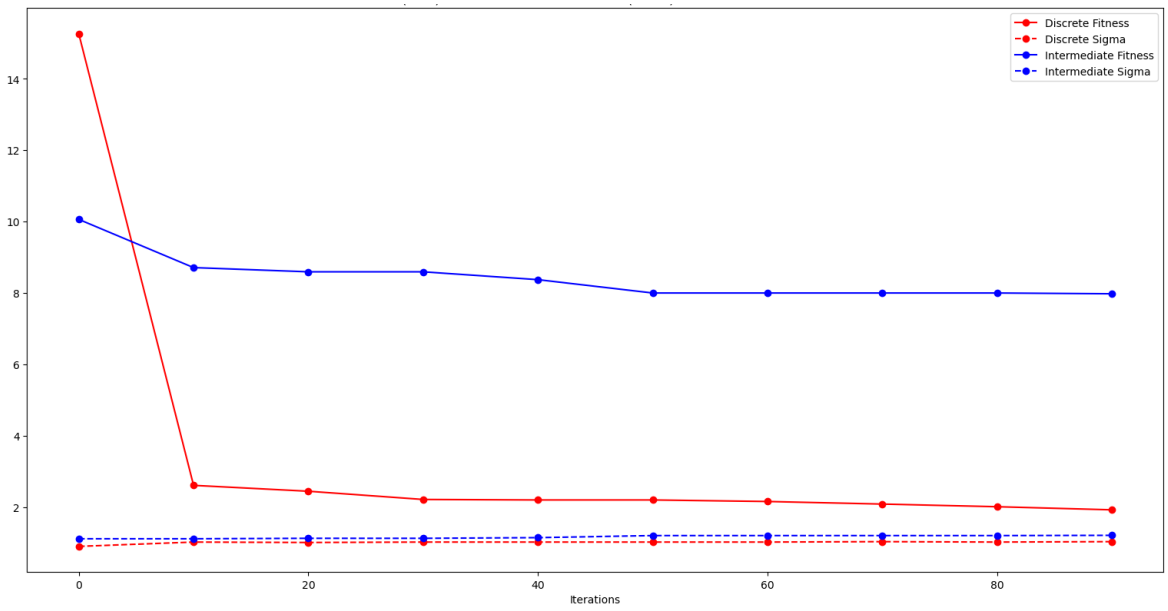

Figure 7: Evolución promedio dimensión 5

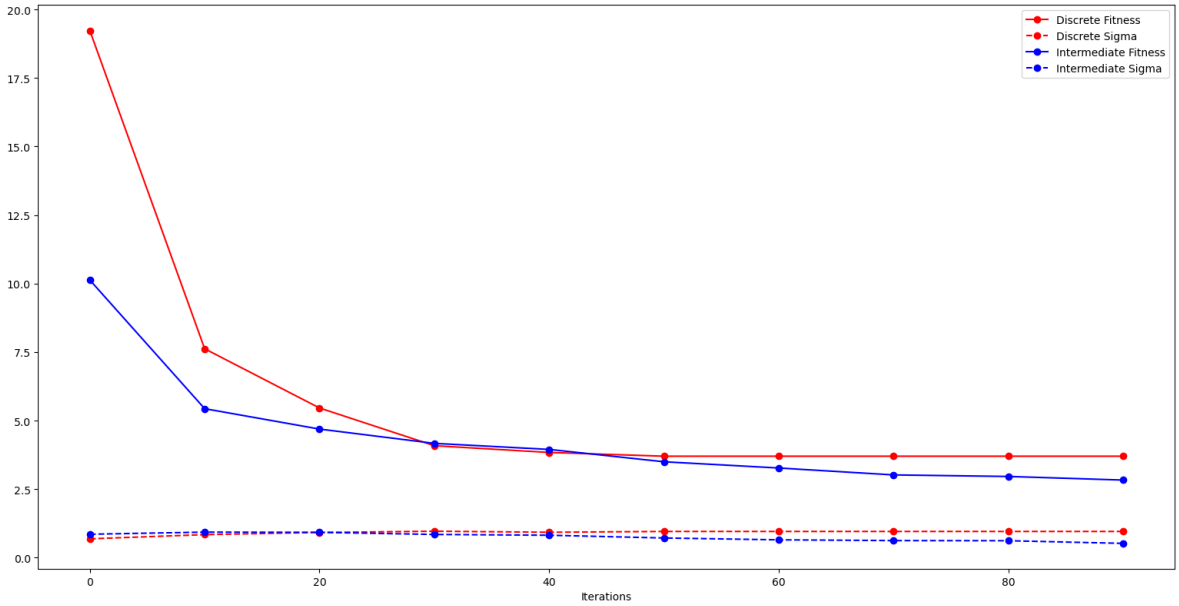

Figure 8: Evolución promedio dimensión 10

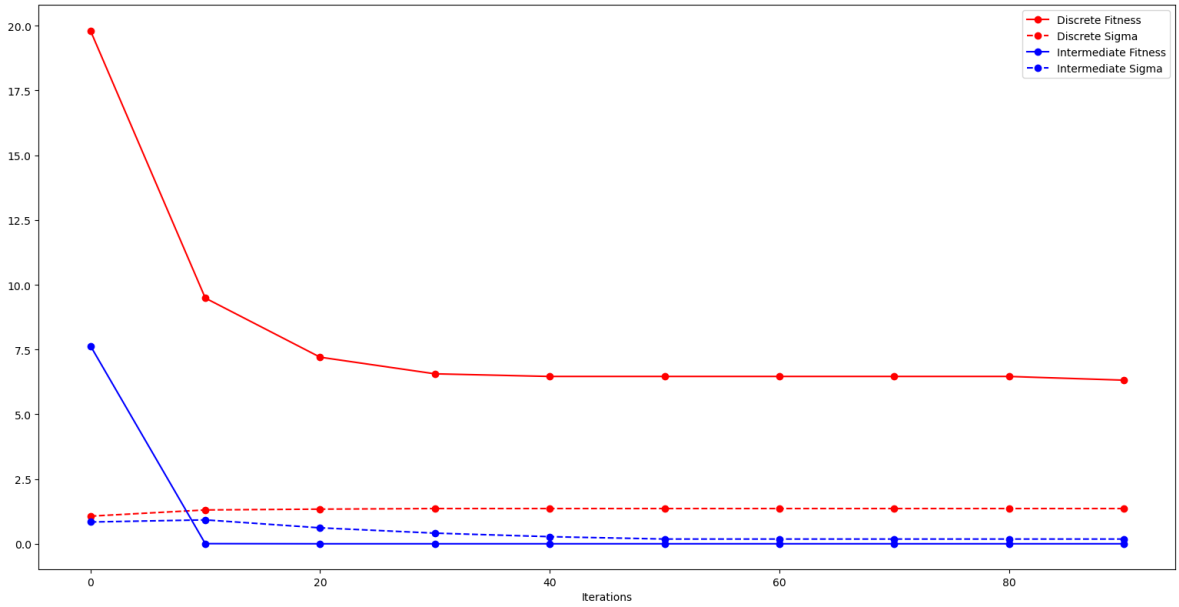

Figure 9: Evolución promedio dimensión 20

El aparente comportamiento después de realizar 10 repeticiones para cada dimensión es que, la recombinación discreta parece encontrar mejores resultados en dimensiones menores a 10. Mientras que a partir de la dimensión 10, la recombinación intermedia parece encontrar mejores resultados en dimensiones superiores e incluso en dimensión 20 , la comparativa entre el operador discreto e intermedio, resulta en que este último es superior por mudho al encontrar el resultado.

Gráficas Boxplot

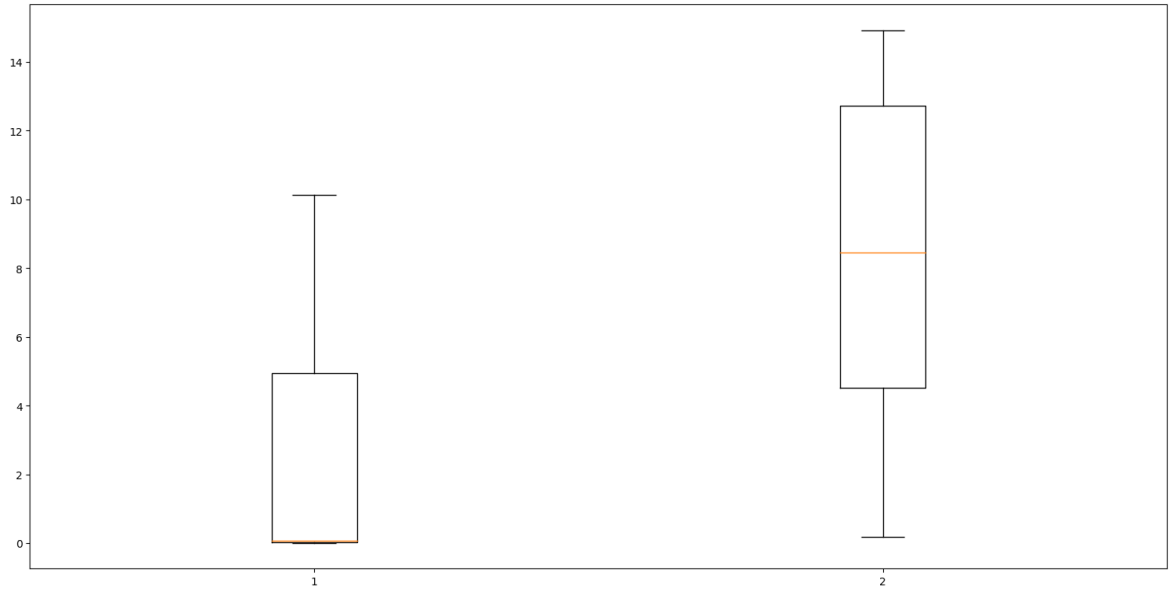

Figure 10: Boxplot dimensión 2 : 1-Discreto, 2-Intermedio

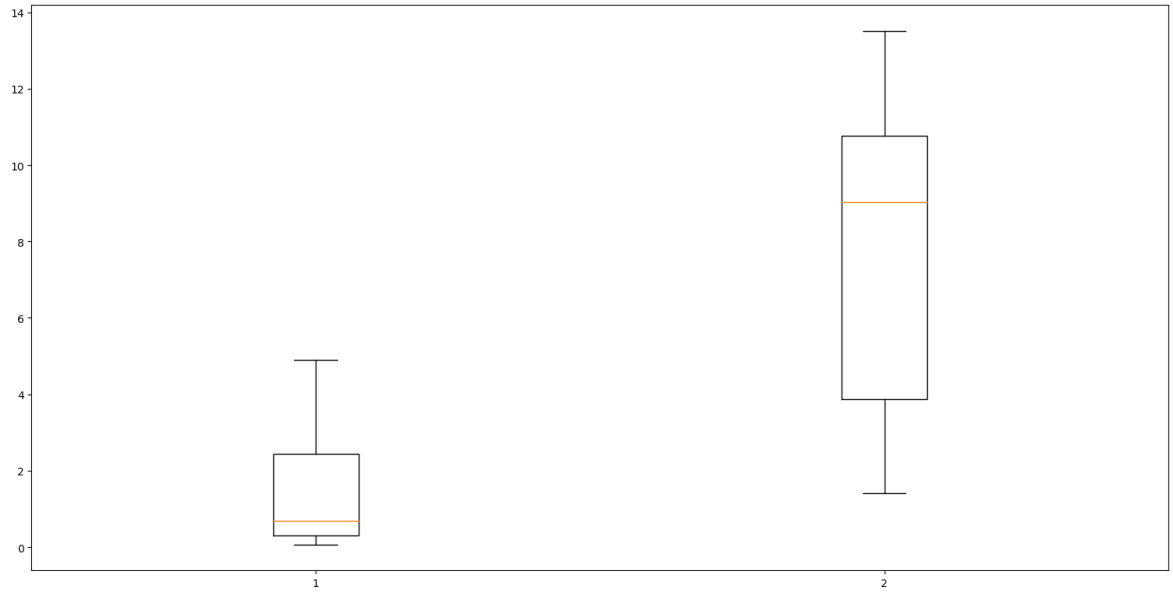

Figure 11: Boxplot dimensión 5 : 1-Discreto, 2-Intermedio

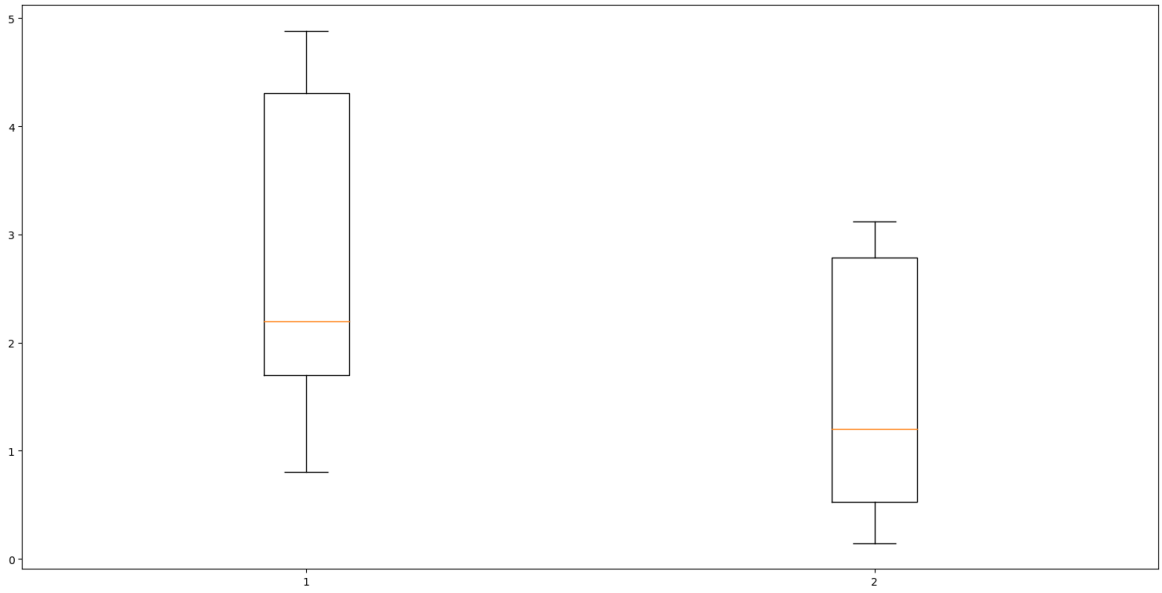

Figure 12: Boxplot dimensión 10 : 1-Discreto, 2-Intermedio

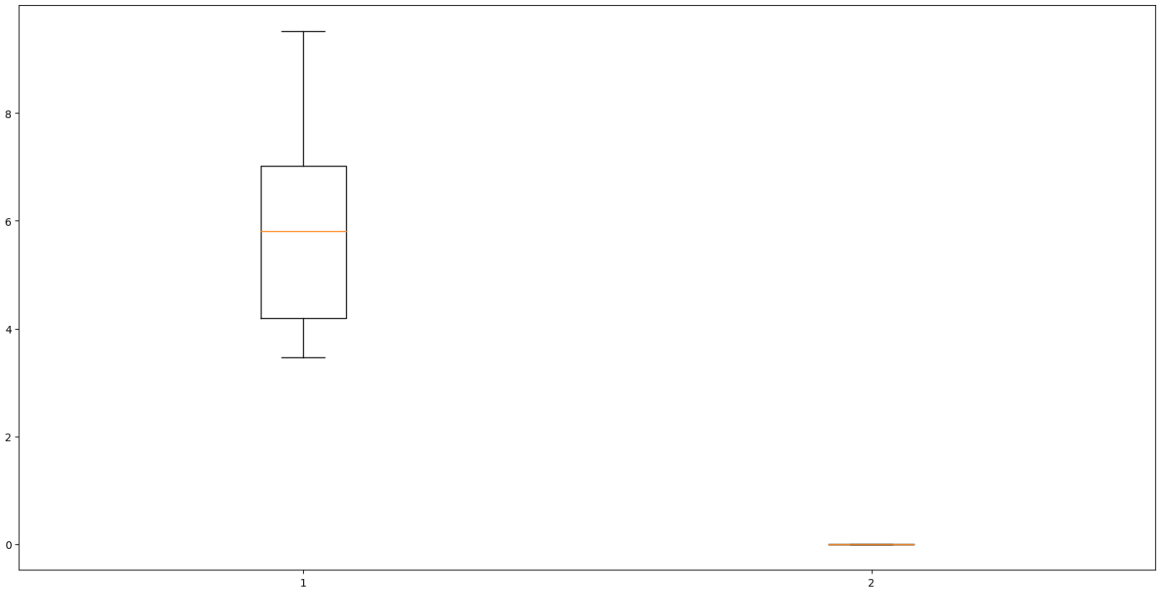

Figure 13: Boxplot dimensión 20 : 1-Discreto, 2-Intermedio

En las gráficas boxplot, la información estadística refleja nuestros comentarios anteriores de la evolución promedio. Parece que a mayor dimensión , mejores resultados obtenemos con recombinación intermedia y a menor dimensión, mejores resultados con recombinación discreta.

Conclusiones

Según Eiben[1] "el esquema de selección generalmente usado en las estrategias de evolución es (μ, λ), el cual es preferido sobre (μ+ λ) por las siguientes razones :

- El operador (μ+λ) descarta todos los padres y entonces puede en principio dejar (pequeños) óptimos locales, lo cual es ventajoso para problemas multimodales.
- Si la función objetivo cambia durante el tiempo de ejecuci 'n, la selección (μ+ λ) preserva soluciones obsoletas, por lo que es menos capaz de seguir el óptimo móvil.
- La selección (μ+ λ) dificulta la autoadaptación, porque parámetros endógenos pueden sobrevivir por un número de generaciones relativamente grande. Por ejemplo, un individuo con variables (vector de valores) relativamente bueno pero con par 'ametros endógenos pobres comúnmente tendrá descendencia cuya función objetivo sea mala. Por lo tanto su descendencia será removida por la política elitista, mientras que los parámetros endógenos obsoletos del padre pueden sobrevivir más tiempo del deseado."

Observaciones
La recombinación discreta considera genes aleatorios para generar descendencia, sin embargo, los genes que considera no son necesariamente los que producen buenos resultados de la función objetivo. Esto le permite tener una alta capacidad de exploración ya que la información de los padres no tiene que ver con los mejores individuos. En espacios de dimensiones menores (2-5) hay menos soluciones posibles por lo que si tenemos mucha exploración , entonces podemos acercarnos al óptimo con mayor facilidad.
La recombinación intermedia tiene una alta probabilidad de considerar a los mejores individuos para generar la información, una vez seleccionados los padres, se realiza el promedio de los valores por lo que la información de los mejores individuos es legada a su descendencia. Esto le permite tener una alta capacidad de explotación ya que la información de la descendencia contiene información de los mejores individuos. En espacios de dimensiones mayores (¿10) el espacio de soluciones es enorme, por lo que es difícil explorar, conviene considerar información de los mejores individuos para acercarse al óptimo.
Sin embargo puede haber un sesgo que no consideramos en las observaciones anteriores, en la experimentación ambos factores utilizan la misma población de posibles padres y población de descendencia. Por lo que la dimensión puede tener una relación importante con la población que se genera, por ejemplo una relación entre dimensión y μ y λ que nos permita obtener una equivalancia respecto al espacio de búsqueda considerado, el cual cambia considerablemente dependendiendo de la dimensión.

References

[1] J.E. Smith A.E. Eiben. *Natural Computing Series : Introduction to Evolutionary Computing.* Springer, Springer Heidelberg New York Dordrecht London, 2 edition, 2015.

[2] Hans-Georg Beyer. *Natural Computing Series : The Theory of Evolution Strategies.* Springer, Springer-Verlag Berlin Heidelberg, 1 edition, 2001.

[3] Hans-Paul Schwefel Hans-Georg Beyer. Evolution strategies - a comprehensive introduction. *Natural Computing*, DOI: 10.1023(A:1015059928466):536, 2002.