Modelos avanzados de IO

Tema 2. Metaheurísticos

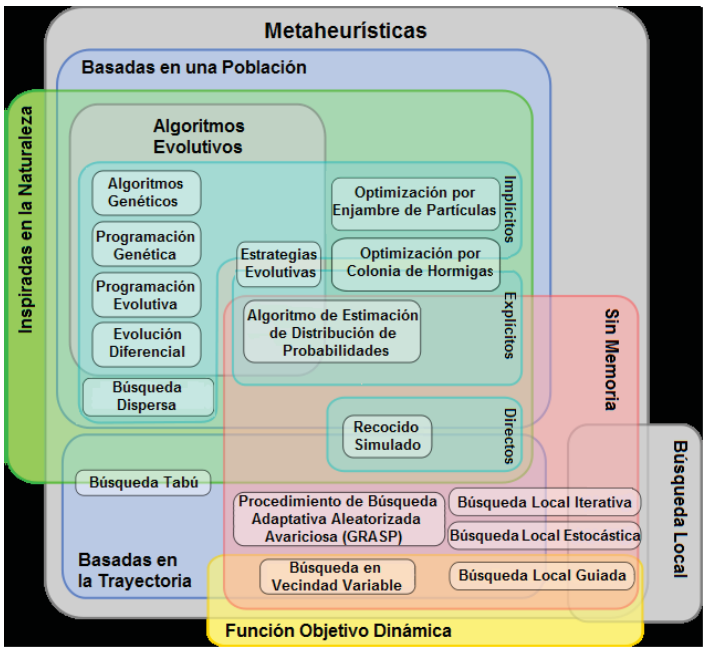
2.1.- Introducción a los Metaheurísticos
2.2.- GRASP
2.3.- Iterated local Search ILS
2.4.- Iterated Greedy IG
2.5.- Algoritmos genéticos GA

2.1.– Introducción a los Metaheurísticos

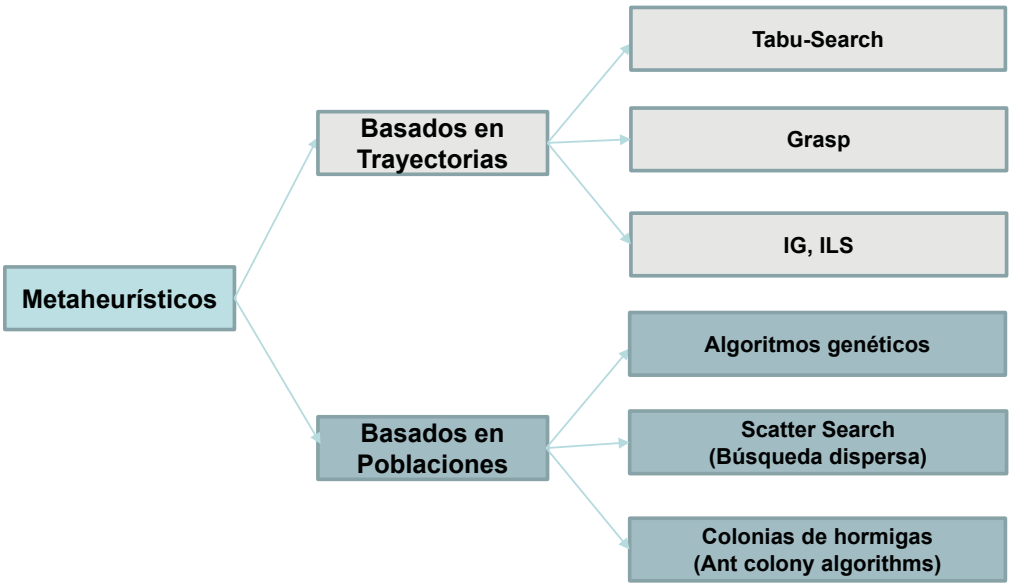
Metaheurísticos

- Procedimientos generales de alto-nivel que coordinan heurísticos y reglas simples para encontrar buenas soluciones de calidad para problemas computacionalmente difíciles.
- Puede aplicarse a un problema concreto de forma sencilla y eficiente sin tener mucho conocimiento acerca del problema.
- Últimas tendencias: Introducir conocimiento del problema. Cuanto más conocimiento más eficiencia pero a costa de perder tanto sencillez como generalidad.
- Frontera difusa entre heurísticos y metaheurísticos.

Metaheurísticos: Clasificación



Metaheurísticos: Clasificación



2.2.– GRASP

GRASP

- Greedy Randomized Adaptive Search Procedure (GRASP) (procedimientos de búsqueda basados en funciones voraces aleatorizadas que se adaptan)
- Feo y Resende (1995) para resolver problemas de cubrimientos de conjuntos.
- Se basa en la premisa de que soluciones iniciales buenas y diversas juegan un papel importante en el éxito de los métodos de búsqueda locales.
- Método multi-arranque en el que cada iteración consiste en dos fases:
  1. **Fase de construcción**: se encarga de construir una solución factible de calidad, usualmente con un heurístico constructivo con aleatoriedad.
  2. **Fase de mejora**: se encarga de mejorar la solución obtenida en la primera fase, usualmente con una búsqueda local.

GRASP: esquema general

Mientras (*Condición de Parada)*
{

Fase constructiva:
Obtener una solución con un algoritmo constructivo aleatorizado: S

Fase de mejora:
Realizar un proceso de búsqueda local a partir de la solución *S*.

Actualización Mejor Solución:
Si la solución obtenida mejora la mejor almacenada: actualizarla.

}

- Este esquema es el más básico, existen múltiples ajustes y mejoras.
- La aleatorización del constructivo se puede escoger más o menos profunda
- La búsqueda local se puede escoger más o menos profunda
- Pruebas computacionales para comprobar, con un tiempo límite, para estudiar qué decisión es la correcta.

GRASP para TSP, mochila, asignación

1. Mientras no se hayan realizado 1000 iteraciones
   1.1. Construir una solución S mediante el constructivo aleatorizado.
   1.2. Aplicar la búsqueda local estudiada.
2. Devolver la mejor solución obtenida en todo el algoritmo.

2.3.– Iterated Local Search ILS

Iterated Local Search

- Comienza con una solución factible. Se puede calcular con un constructivo o que sea la mejor de varias soluciones factibles obtenidas con un constructivo aleatorizado.
- En cada iteración, se perturba la solución actual.
  - Con esta perturbación se calcula una nueva solución factible 'diferente' de la actual. En general será de peor calidad que la original.
  - La clave es que la nueva solución no sea muy parecida a la solución actual (caeríamos en el mismo óptimo local) ni muy diferente (sería como comenzar con una solución aleatoria). De ahí que el cambio deba ser del 30-40% de la solución (aunque depende).
- A continuación se aplica una búsqueda local a la nueva solución. Si esta nueva solución es mejor que la solución actual, nos quedamos con esta solución para la iteración siguiente. En caso contrario, seguimos con la misma solución.

Iterated Local Search

Para un problema de minimizar, el algoritmo sería el siguiente.

1. Obtener una solución inicial S.
2. Mientras no se cumpla el criterio de parada
   2.1 S' = Perturbación(S).
   2.2. S'' = Búsqueda_Local(S').
   2.3. Si (f(S'') < f(S), entonces S = S''.

Luego habría que especificar cada elemento del algoritmo.

El criterio de parada puede ser:
- Un número total de iteraciones.
- Un cierto número de iteraciones sin mejorar la solución.
- Un tiempo máximo.

Bibliografía

Lourenço, H.R.; Martin O.; Stützle T. (2003). "Iterated Local Search". *Handbook of Metaheuristics*. Kluwer Academic Publishers, International Series in Operations Research & Management Science. **57**: 321–353.

Iterated Local Search

Ejemplo: Problema de la mochila

Codificación: Vector de variables binarias.

Perturbación

1. Escogemos un 50% de los proyectos.
2. De esos proyectos, los 0's los transformamos a 1's y los 1's a 0's.
3. Si la solución es infactible, ordenamos los proyectos en orden creciente de beneficio.
4. Los vamos eliminando hasta que la solución sea factible.
5. Escogemos aleatoriamente un orden* de entre todos los proyectos no seleccionados y en ese orden los añadimos si la solución sale factible.

Búsqueda Local: Algunas de las estudiadas.

* Puede ser por ejemplo orden decreciente de beneficio.

2.4.– Iterated Greedy IG

Iterated Greedy

- Es similar al ILS. Este metaheurístico se basa en disponer de un algoritmo constructivo 'bueno', que proporciones buenas soluciones si la elección de los elementos a añadir es la correcta.
- Se comienza aplicando el constructivo 1 o el aleatorizado varias veces hasta obtener una solución inicial S.
- A continuación, en cada iteración se aplica lo siguiente.
  - Se destruye parte de la solución.
  - Se aplica el algoritmo constructivo a la parte que se ha destruido.
  - Se aplica una búsqueda local a la nueva solución.
  - Se decide si aceptar la nueva solución o no.

Iterated Greedy

Para un problema de minimizar, el algoritmo sería el siguiente.

1. Obtener una solución inicial S.
2. Mientras no se cumpla el criterio de parada
   2.1 S' = Destrucción(S).
   2.2. S'' = Construcción(S').
   2.3. S''' = Búsqueda_Local(S'').
   2.4. Estudiar si reemplazar S por S'''.

Luego habría que especificar cada elemento del algoritmo.

El criterio de parada puede ser el mismo que en IG

Bibliografía
Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (3): 2033-2049

Iterated Greedy

Ejemplo: Problema de la mochila

Codificación: Vector de variables binarias.

Constructivo: Ordenar los proyectos en orden decreciente de beneficio (de mayor a menor). Incluirlos en la solución en ese orden si la solución es factible.

Destrucción: Fijar un 50% de los proyectos (tanto 0's como 1's). Al resto de proyectos le aplicamos el constructivo. Es decir, se ordenan en orden decreciente de beneficio. Se incluyen en la solución en ese orden si la solución es factible.

Solución inicial: (0 1 1 1 1 0 0). Fijamos: 4,5,1. Ordenamos el resto en orden decreciente del beneficio, 7 2 6 3. al intentar incluirnos nos salen los proyectos seleccionados 4,5,6,7, la solución (0 0 0 1 1 1 1).

Búsqueda Local: Algunas de las estudiadas.

2.5.– Algoritmos Genéticos GA

Evolución Natural

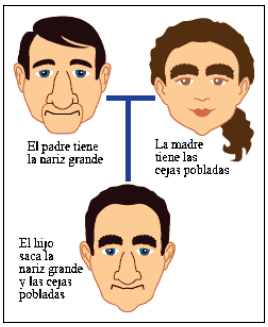
- Introducidos por Johh Holland a mitad de los 70.
- Inspirados en el proceso de evolución de las especies formuladas por Charles Darwin a mediados del siglo XIX.

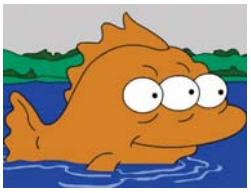
  Los individuos que tienen una mejor adaptación al medio se caracterizan por tener una probabilidad más elevada de vivir más tiempo, con lo que tendrán más posibilidades de generar descendencia que hereden sus buenas características.
  En cambio, los individuos con peor adaptación al medio, tienen menos probabilidad de sobrevivir, por lo que tendrán menos oportunidades de generar descendencia y probablemente acaben extinguiéndose.

- Utilizan el principio de selección natural para resolver problemas de optimización "complicados".

Evolución Natural

- Cada descendiente hereda algunos de los cromosomas de cada uno de los padres, donde los genes de los cromosomas determinan las características individuales de los hijos.
- Un hijo que hereda las mejores características tiene una mayor posibilidad de sobrevivir y se convierte en padre pasando estas características a la siguiente generación.
- La población tiende a mejorar de manera lenta a lo largo del tiempo por medio de este proceso.
- De vez en cuando ocurre una mutación aleatoria que cambia las características del cromosoma heredado. Los hijos con mutaciones deseables tienen una mayor posibilidad de sobrevivir y mejorar la especie.
- La evolución opera en los cromosomas en lugar de en los individuos.





Transfiriendo la evolución a los problemas de optimización

- Cromosomas → Codificación de la solución
- Conjunto de individuos de una población → conjunto de soluciones de un problema.
- Calidad del cromosoma (fitness) → Función objetivo
- Mecanismo de supervivencia de mejores individuos → Seleccionamos las mejores soluciones
- Reproducción y Mutación → Mecanismos que se crean para operar sobre las codificaciones de las soluciones y crear nuevas soluciones.
- **Elitismo**: en la naturaleza los "más aptos" se reproducen más, viven más etc. Tendremos algoritmos con más elitismo o menos elitismo, dependiendo de la probabilidad de supervivencia y de reproducción que le demos a las soluciones peores en fitness. Un GA tiene que tener una componente grande de elitismo.

Esquema Básico de un GA

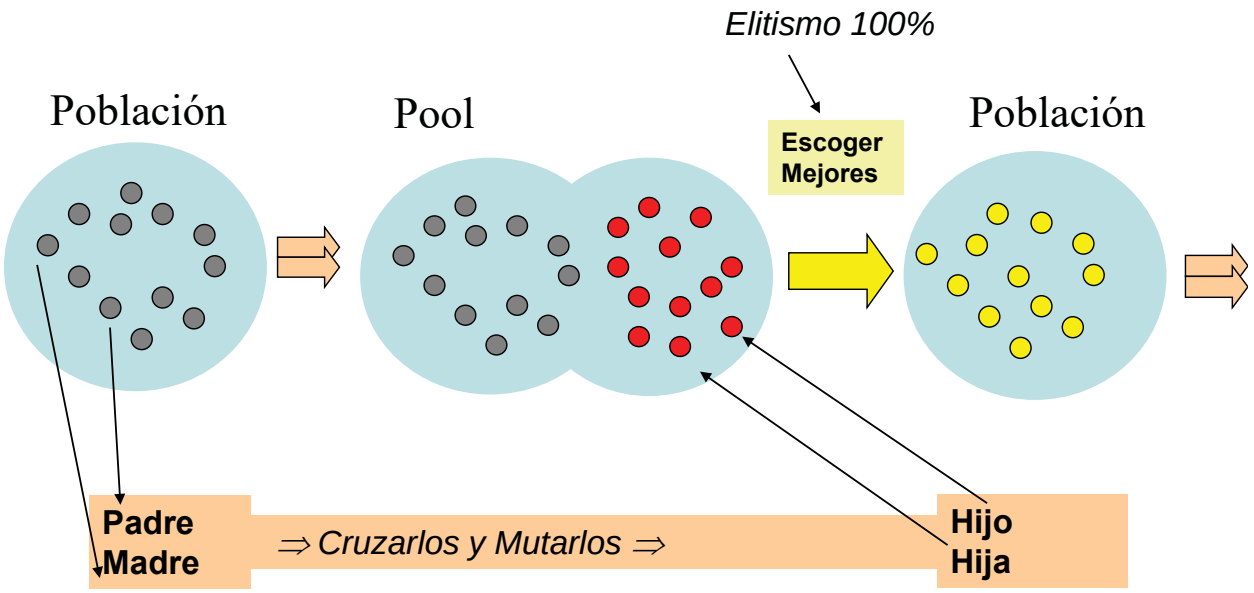
1. Generar Población inicial P.
2. Mientras no se cumpla un criterio de parada:
    - 2.1. H = ∅
    - 2.2. Realizar m veces (|P|/2 veces):
        - 2.2.1. Seleccionar 2 progenitores de P con el método de la ruleta (a mejor **fitness** mayor probabilidad).
        - 2.2.2. Aplicarles el **mecanismo de cruce**, obteniendo 2 hijos.
        - 2.2.3. Aplicarles la **mutación** y después evaluarlos.
        - 2.2.4. Añadir los hijos al conjunto H.
    - 2.3. Calcular la nueva población P a partir de la antigua P y de H.
3. Ofrecer la mejor solución y su función objetivo.

Criterio de Parada:

- Fijar un número máximo de soluciones generadas o de tiempo
- Fijar un número máximo de iteraciones sin mejorar
- Fijar un límite de tiempo

Aplicamos elitismo en:
- Iteración 2.2.1
- Iteración 1?
- Iteración 2.3?

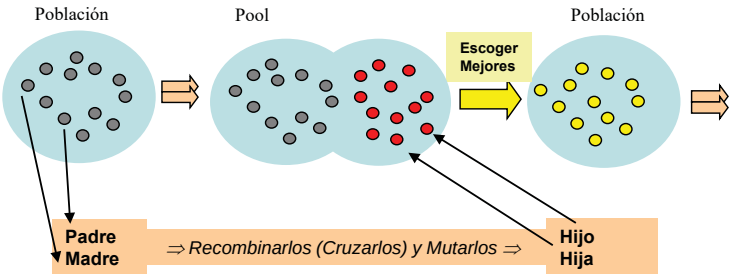
Esquema Básico de un GA



Elementos de un GA

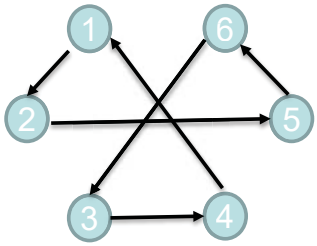
Elementos principales de un algoritmo genético:

1. Representación de las soluciones del problema (Codificación).
2. Función de evaluación (fitness/aptitud).
3. Mecanismo de creación de la población inicial.
4. Operadores genéticos (cruce & mutación).
5. Selección de padres. Mecanismo de selección [probabilística].
6. Selección de supervivientes para la siguiente población. Reducción de la población.
7. Valores para los distintos parámetros del algoritmo: Criterio de parada.



Elementos de un GA
Codificación: Permutación

- Los individuos se representan como permutaciones
- Se utilizan principalmente en problemas de secuenciación o de rutas (ordenación) como el TSP.
- Necesitan operadores de cruce y mutación especiales para garantizar que el resultado de aplicar un operador sigue siendo una permutación



(1,2,5,6,3,4) → CROMOSOMA

Elementos de un GA
Codificación: Representación Binaria de Enteros

Representación BINARIA de números enteros

Max $f(x)=12x^5-975x^4+28000x^3+1800000x$
s.a. $0 \leq x \leq 200$, x entero

GENOTIPO ↑ FENOTIPO ↑

(1,0,1,0,0,0,1,1) = 163

$1 \times 2^7+0 \times 2^6+1 \times 2^5+0 \times 2^4+0 \times 2^3+0 \times 2^2+1 \times 2^1+1 \times 2^0 = 163$

- Si hubiera más variables, cada variable tendría esa representación.
- Formas análogas para números reales.

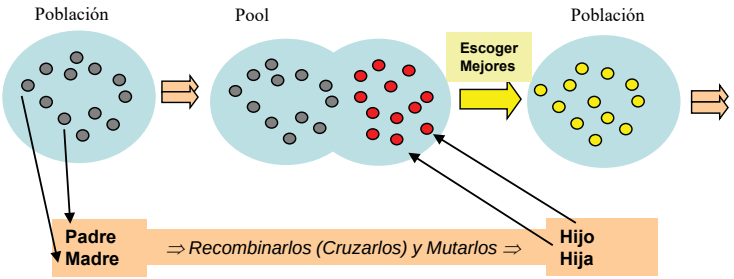
Elementos de un GA

1. Representación de las soluciones del problema (Codificación).
2. Función de evaluación (fitness/aptitud).
3. Mecanismo de creación de la población inicial.
4. Operadores genéticos (cruce & mutación).
5. Selección de padres. Mecanismo de selección [probabilística].
6. Selección de supervivientes para la siguiente población. Reducción de la población.
7. Valores para los distintos parámetros del algoritmo: Criterio de parada.

- Utilizamos constructivo aleatorizado, inteligente o no. Cuanta + aleatoriedad menos elitismo.
- El tamaño de la población es un parámetro a determinar mediante pruebas
  - Poblaciones pequeñas: Riesgo de no cubrir el espacio de búsqueda. Riesgo de no encontrar el óptimo. Sólo para mucho elitismo
  - Poblaciones muy grandes : GA muy lento. Para poco/nada elitismo.

Elementos de un GA

1. Representación de las soluciones del problema (Codificación).
2. Función de evaluación (fitness/aptitud).
3. Mecanismo de creación de la población inicial.
4. Operadores genéticos (cruce & mutación).
5. Selección de padres. Mecanismo de selección [probabilística].
6. Selección de supervivientes para la siguiente población. Reducción de la población.
7. Valores para los distintos parámetros del algoritmo: Criterio de parada.

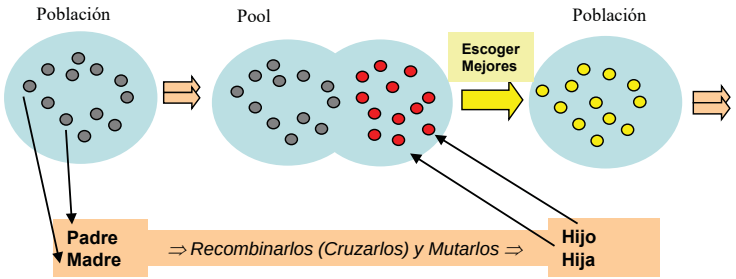


Elementos de un GA
Mecanismos de cruce y mutación

- Nos permiten generar nuevos individuos (soluciones)
- La selección de mecanismos concretos depende de la codificación.
- **Cruce**: consiste utilizar los genes de dos progenitores para generar un nuevo individuo hijo. Representa la reproducción de los individuos.
- **Mutación**: consiste en el cambio aleatorio de parte de un individuo. La mutación se emplea como mecanismo para preservar la diversidad de las soluciones y explorar nuevas zonas del espacio de soluciones.
- En algunos problemas, después del cruce y la mutación podemos tener que aplicar un mecanismo para obtener factibilidad o para no trabajar con soluciones parciales.

Elementos de un GA

1. Representación de las soluciones del problema (Codificación).
2. Función de evaluación (fitness/aptitud).
3. Mecanismo de creación de la población inicial.
4. Operadores genéticos (cruce & mutación).
5. Selección de padres. Mecanismo de selección [probabilística].
6. Selección de supervivientes para la siguiente población. Reducción de la población.
7. Valores para los distintos parámetros del algoritmo: Criterio de parada.



Elementos de un GA
Reducción de Población

Reducción de la población/Selección de supervivientes/Reemplazo:

La mayoría de los algoritmos evolutivos utilizan una población de tamaño fijo, por lo que hay que definir una forma de crear la siguiente población a partir del conjunto de individuos disponibles (padres+hijos).

Podemos seleccionar padres, obtener hijos y directamente estudiar si introducirlos en la población o construir una población de hijos antes de decidir cuáles introducir en la siguiente iteración

Técnicas:

- Reducción completamente elitista: Se escogen los mejores del conjunto de padres e hijos. Un GA suele requerir de elitismo para converger al óptimo.
- Probabilísticos. Los de la siguiente población se escogen en función de la función de evaluación. Cuanto mayor función de evaluación mayor probabilidad de ser escogido.
- Aleatorios. Escoger un cierto número entre los mejores de padres e hijos y el resto aleatoriamente de entre los hijos no seleccionados.

Elementos de un GA. TSP
Operador de cruce para permutaciones

Para Permutaciones - Cruce de orden:

Mantiene un orden de un sub-tour de un padre y mantiene el orden relativo del resto de ciudades del otro padre

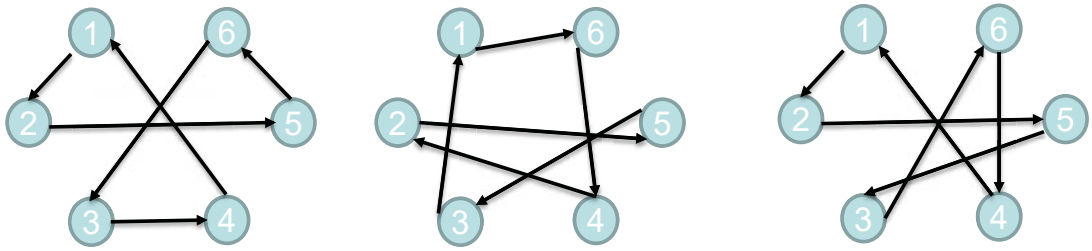
- Seleccionar una parte arbitraria del primer padre.
- Copiar esta parte en el hijo
- Copiar los números que no están en la primera parte:

  Empezando desde el punto de cruce

  Utilizando el orden en el que aparecen en el 2 padre

  Volviendo al principio del cromosoma cando hayamos llegado al final
- El segundo hijo se crea intercambiando los papeles de los padres

Elementos de un GA. TSP
Operador de cruce para permutaciones



Padre: (1,2,5,| 6,3,4)

Madre: (3,1,6,| 4,2,5)

Construcción Hijo:
- Fijamos parte Padre: (1,2,5,x,x,x)
- Faltan: 3,4,6. Los completamos según orden Madre: (1,2,5,3,6,4)

Construcción Hija:
- Fijamos parte Madre: (3,1,6,x,x,x,x)
- Faltan: 2,4,5. Los completamos según orden Padre: (3,1,6,2,5,4)

Elementos de un GA. Mochila
Operador de cruce para representación binaria

Cruce en un punto (1-point crossover)

Se selecciona un punto de cruce aleatoriamente

Se dividen los padres en ese punto

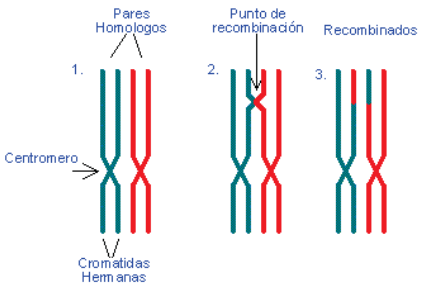
Se crean hijos intercambiando partes de los cromosomas

Padres: (0,0,0,|0,0,0,0,0,0,0,0)

(1,1,1,|1,1,1,1,1,1,1,1)

Hijos: (0,0,0,|1,1,1,1,1,1,1,1)

(1,1,1,|0,0,0,0,0,0,0,0)



Elementos de un GA. Mochila
Operador de cruce para representación binaria

Cruce uniforme

Se elige al azar el padre del que proviene cada gen para el primer hijo (del segundo hijo es el inverso)

Padres: (0,0,0,0,0,0,0,0,0,0)

(1,1,1,1,1,1,1,1,1,1)

Hijos: (0,1,0,1,1,0,1,1,0,1)

(1,0,1,0,0,1,0,0,1,0)

Elementos de un GA. Mochila.
Operador de mutación.

Para permutaciones
Inserción: Elegir dos genes aleatoriamente y colocar el segundo justo después del primero
(1,2,3,4,5,6) → (1,2,5,3,4,6)

Intercambio: Seleccionar dos genes aleatoriamente e intercambiarlos
(1,2,3,4,5,6) → (1,5, 3,4,2,6)

Para representación binaria
Dado un cromosoma, se altera el valor de cada gen con una cierta probabilidad

Padre: (1,1,1,1,0,0,0,1,0)

Hijo: (1,0,1,1,1,0,0,1,1)

Elementos de un GA. Mochila.
Mecanismos para factibilidad

En algunos problemas, después de los mecanismos de cruce y mutación pueden suceder dos consecuencias negativas:

- La solución es infactible, por ejemplo (1 1 1 1 1 1 1).
- La solución está incompleta (desaprovecha recursos), por ejemplo (0 0 0 0 0 0 0).

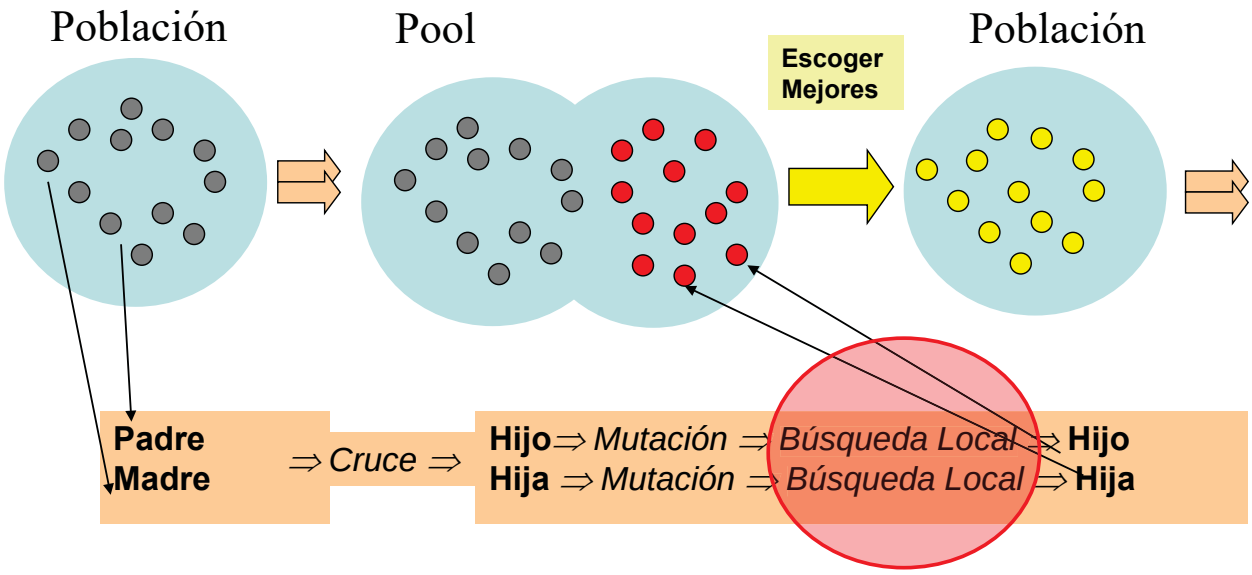
En el 1er caso, aplicaremos un mecanismo para conseguir factibilidad.

1. Ordenar los proyectos asignados en orden creciente de beneficio.
2. Ir eliminando los proyectos en ese orden hasta que la solución sea factible.

En el 2º caso, o después de aplicar el mecanismo de factibilidad, aplicamos el siguiente mecanismo (básicamente es el constructivo inteligente aplicado a la solución parcial obtenida).

1. Ordenar los proyectos no asignados en orden decreciente de beneficio.
2. Ir añadiendo los proyectos en ese orden mientras la solución sea factible.

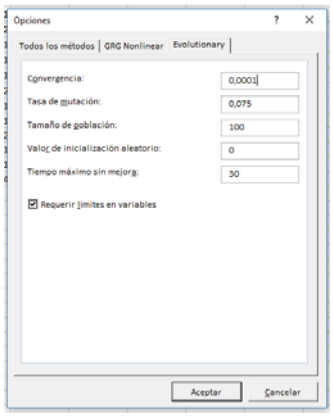
GA + Búsquedas Locales



EVOLUTIVO DEL EXCEL

- Basado en la genética, evolución y supervivencia del más apto.
- Ventajas:
  - La complejidad de la función objetivo no le afecta.
  - La complejidad de las restricciones no le afecta.
  - Como evalúa muchas soluciones: evita quedar atrapado en óptimos locales.
- Desventajas:
  - Puede tardar más tiempo que el solver estándar.
  - No funciona bien con problemas que tienen muchas restricciones.
  - Como es un proceso aleatorio pueden dar resultados distintos en sucesivas ejecuciones.
  - La solución encontrada no suele ser la óptima (aunque suele estar muy cerca de serlo).

EVOLUTIVO DEL EXCEL



Convergence

•In the Convergence box, type the maximum percentage difference in objective values for the top 99% of the population that Solver should allow in order to stop with the message “Solver converged to the current solution.” Smaller values here normally mean that Solver will take more time, but will stop at a point closer to the optimal solution.

Mutation Rate

•In the Mutation Rate box, type a number between 0 and 1, the relative frequency with which some member of the population will be altered or “mutated” to create a new trial solution, during each “generation” or subproblem considered by the Evolutionary method. A higher Mutation Rate increases the diversity of the population and the chance that a new, better solution will be found; but this may increase total solution time.

Population Size

•In the Population Size box, type the number of different points (values for the decision variables) you want the Evolutionary method to maintain at any given time in its population of candidate solutions. The minimum population size is 10 members; if you supply a value less than 10 in this box, or leave it blank, the Evolutionary Solver uses a population size of 10 times the number of decision variables in the problem, but no more than 200.

Random Seed

•In the Random Seed box, type a positive integer number to be used as the (fixed) seed for the random number generator used for a variety of random choices in the Evolutionary method. If you enter a number here, the Evolutionary method will use the same choices each time you click Solve. If you leave this box blank, the random number generator will use a *different* seed each time you click Solve, which may yield a different (better or worse) final solution.

Maximum Time without Improvement

•In the Maximum Time without Improvement box, type the maximum number of seconds you want the Evolutionary method to continue without a meaningful improvement in the objective value of the best solution in the population, before it stops with the message “Solver cannot improve the current solution.

Require Bounds on Variables

•Select the Require Bounds on Variables check box to specify that the Evolutionary method should run only if you have defined lower and upper bounds on all decision variables in the Constraints list box. The Evolutionary method is far more effective if you define bounds on all variables; the tighter the bounds on the variables that you can specify, the better the Evolutionary method is likely to perform.

BIBLIOGRAFÍA

- Hillier, F. S., Lieberman, G.J., Investigación de operaciones, 10ª edición, 2015.
- Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. Matematiques, vol. 1, No 1, p. 3-62.
- Resende, M.G.C., Ribeiro, C.C. (2016), Optimization by GRASP, Greedy Randomized Adaptive Search Procedures, Springer.
- Taha, H. A. Investigación de operaciones. Pearson, 9ª edición , 2012.
- Burke, E.K., Kendall, G., Search Methodologies: Introductory Tutorials in Optimization and Decision, 2016, Springer