

Computación evolutiva: primera práctica

Andrés Mañas Mañas

December 7, 2016

Abstract

Implementación desde cero un algoritmo genético para la resolución del Problema de la Mochila Binario y evaluación de los resultados con herramientas gráficas de monitoreo.

La implementación cumple con estas características: representación binaria de los individuos, función fitness basada en ordenamiento por ratio de los objetos y de forma que trate de forma adecuada aquellos individuos que representan soluciones no factibles al problema de la mochila binario, población inicial aleatoria, selección de padres por torneo, cruce por punto con probabilidad dada, mutación de todos los genes con una probabilidad dada, selección de supervivientes atendiendo a un modelo generacional con elitismo.

En la implementación, todos los metaparámetros del algoritmo son configurables al lanzarse.

1 Contenidos

1. Contenidos
2. Introduccion
 1. MUY IMPORTANTE !!!
3. Descripción del problema y requisitos
4. Implementación
 1. Representación del conocimiento
 2. Funciones más relevantes
5. Formas de ejecutar el algoritmo
 1. Ejecución desde consola
 2. Ejecución programática del algoritmo
 3. Ejecución desde fichero del algoritmo
6. Evaluación Experimental
 1. Análisis de resultados
 2. Experimento simple
 1. Curvas de evolución
 2. Conclusiones del experimento simple
 3. Experimento complejo
 1. Curvas de evolución
 2. Conclusiones del experimento complejo

2 Introduccion

El presente documento contiene el informe con el trabajo realizado para la primera práctica de Computación Evolutiva por el alumno Andrés Mañas Mañas.

El lenguaje de programación elegido para la realización de la práctica es **Clojure** (un dialecto de Lisp). He elegido este lenguaje porque hace muy sencilla la representación del conocimiento como estructuras de datos muy fácilmente manipulables. Es además mi lenguaje favorito.

En el material entregado para esta práctica se incluye el archivo **README.pdf**. Es del todo recomendable leer tal archivo en primer lugar para:

- documentarse sobre cómo disponer de un entorno en el que ejecutar el código fuente de la práctica
- visualizar rápidamente la estructura de árbol con todos los documentos que se incluyen en la práctica

Parte de los entregables de la práctica consisten en un sistema que permita ver la evolución del algoritmo y los resultados alcanzados. No son tantas las librerías que permiten visualizar gráficos en Clojure. Si acaso la más destacada sea <http://incanter.org/>. Sin embargo es una librería pesadísima, y he preferido no agregarla como dependencia del proyecto. Además de lo desagradable que es que al ejecutar los algoritmos se vayan abriendo popups en background.

Por eso me he tomado la libertad de desplegar en un pequeño servidor que tengo en

AWS un web service en php y un panel html que permite ver en tiempo real la evolución de una ejecución del algoritmo de aprendizaje.

Cuando se lanza una ejecución de un experimento, de forma transparente se envían periódicamente estadísticas del avance del algoritmo a <http://amanas.ml/ce/service.php>

Por otro lado, si simultáneamente tenemos abierta en un navegador la página <http://amanas.ml/ce/status.html>, podremos ir viendo en tiempo real y de forma gráfica la evolución del aprendizaje del experimento que estemos ejecutando en cada momento.

2.1 MUY IMPORTANTE !!!

Hay que disponer de conexión a internet y abrir <http://amanas.ml/ce/status.html> cuando ustedes vayan a probar los algoritmos que les entrego con estos materiales.

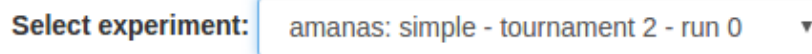


Figura 1. Selector de experimentos

Cada vez que se inicia un experimento, uno de los parámetros que se indican en la configuración del mismo es el nombre del experimento. Después, en la página <http://amanas.ml/ce/status.html> basta seleccionar del combo box el nombre que se ha indicado en la configuración y así se puede ver el resultado de la ejecución del algoritmo.

Estos informes gráficos tienen configurado un tiempo de vida en el servidor que debiera ser suficiente para que siguieran estando disponibles cuando ustedes decidan visualizarlos. En el caso de todos los realizados por mí, el nombre siempre lo empiezo por “amanas: ...”. De este modo, si ustedes realizan experimentos de los mismo ejemplos que yo he realizado, pueden comparar las distintas realizaciones.

3 Descripción del problema y requisitos

En la primera práctica se nos pide que implementemos desde cero un algoritmo genético para la resolución del **Problema de la Mochila Binario**.

El **Problema de la Mochila Binario** se describe así:

Dada una mochila con cierta “capacidad” y varios objetos con cierto “volumen” y “valor”, se trata de determinar qué objetos hay que introducir en la mochila para maximizar el valor total de los objetos contenidos en la misma. Obviamente, se debe cumplir que la suma de los volúmenes de los objetos introducidos en la mochila no exceda la capacidad de ésta.

Los requisitos pedidos son:

- representación binaria de los individuos

- función fitness basada en ordenamiento por ratio de los objetos y de forma que trate de forma adecuada aquellos individuos que representan soluciones no factibles al problema de la mochila binario
- población inicial aleatoria
- selección de padres por torneo
- cruce por punto con probabilidad dada
- mutación de todos los genes con una probabilidad dada
- selección de supervivientes atendiendo a un modelo generacional con elitismo

4 Implementación

Como dije más arriba, el lenguaje que he seleccionado es **Clojure**, un dialecto de Lisp que se engloba por lo tanto en el paradigma de programación funcional.

A mi juicio, las ventajas de utilizar un lenguaje como Clojure son:

- facilidad en la representación del conocimiento, parte clave del problema
- sencillez en la programación de las distintas funciones clave en el algoritmo de aprendizaje. Es un lenguaje de alto nivel que permite realizar operaciones de mapeo o filtrado con una simple función, por lo que se ahorra mucho código y por lo tanto se evita complejidad en la solución desde la base
- evitar los problemas relacionados con la concurrencia y la mutación del estado de los objetos en el paradigma de la orientación a objetos
- y por qué no, diversión garantizada pues Clojure es el lenguaje más divertido que conozco, y puestos a trabajar, mejor hacerlo con alegría

La única desventaja que le veo a la utilización de clojure es que no es el lenguaje que mejor rendimiento ofrece. Corre sobre la máquina virtual java (aunque también hay intérpretes para .Net o javascript). Y, aunque dependiendo del caso puede ofrecer rendimientos muy superiores a, por ejemplo, java, cuando la carga de computación es elevada suele flaquear un poco. En este sentido, lenguajes como C seguro que brindarían rendimientos mejores.

La implementación del algoritmo que satisface las especificaciones indicadas anteriormente pueden ustedes encontrarlas en el archivo **src/ce/p1.clj**

Vamos a comentar un poco las partes claves del código que entrego.

4.1 Representación del conocimiento

Tenemos que representar como mínimo dos conceptos clave: los objetos (genes) y los individuos.

Cada objeto lo represento como un mapa (estructura clave-valor) con tres claves: nombre del objeto, valor del objeto y volumen del objeto.

```
{:nam "Objeto 0" :val 1 :vol 3}
```

Como parte del preprocesado inicial de los objetos de entrada del problema, agrego automáticamente una nueva propiedad a cada objeto que será el **ratio (valor/volumen)**, de modo

que en los distintos accesos que haga al objeto no tenga que recalcularse este valor y así ahorro en computación. Es decir, quedaría:

```
{:nam "Objeto 0" :val 1 :vol 3 :ratio 1/3}
```

Ésta sería la representación de cada objeto individualmente.

Pero además, en las primeras fases del algoritmo, lo que hago es **ordenar todos los objetos en un array por orden de ratio**. Es decir, los objetos con mayor valor y menor volumen los primeros y los de menor valor y mayor volumen los últimos. La razón de esta ordenación es mejorar el rendimiento de los cálculos que se realizarán para determinar el valor de fitness de cada individuo a lo largo de la evolución.

De este modo, **el conjunto de objetos que tenemos** para meter en la mochila **acabaría representado** de un modo similar al siguiente:

```
[{:nam "Objeto 9" :val 10 :vol 3 :ratio 10/3}
 {:nam "Objeto 5" :val 7 :vol 3 :ratio 7/3}
 {:nam "Objeto 2" :val 8 :vol 4 :ratio 2}
 ...]
```

Tenemos que determinar además cómo representar a los individuos. Siguiendo las indicaciones decido **representar cada individuo como un vector de genes binarios**. Es decir, si el gen n -ésimo de un individuo es `True`, con $0 \leq n < \text{número de objetos}$, entonces ese individuo incluiría en la mochila el objeto n -ésimo. Y no lo incluiría si el gen hubiera tenido el valor `False`. Nótese que en lugar de $[0, 1]$ utilizo $[\text{True}, \text{False}]$, sencillamente porque se me antoja más claro.

Por ejemplo, el individuo `[True False True...]` representa meter en la mochila el primer y tercer objetos (una vez reordenados por ratio, según se comentó anteriormente).

4.2 Funciones más relevantes

```
;; Genera un objeto a partir de sus propiedades.
;; - nam: nombre del objeto
;; - val: valor del objeto
;; - vol: volumen del objeto
(defn new-object [nam val vol]

;; Genera un objeto aleatoriamente con valor entre 1 y max-val y
;; volumen entre 1 y max-vol.
;; - i: número de objeto
;; - max-val: valor máximo
;; - max-vol: volumen máximo
(defn rand-object [i max-val max-vol]

;; Necesitamos una función que genere una estructura con los objetos iniciales
;; del problema optimizada para accesos y para economía de computación.
;; Para ello:
;; 1. enriquezco los objetos iniciales con el valor del ratio
;; (val/vol) para no tener que recalcularlo en cada acceso.
```

```
;; 2. ordeno en un array el conjunto resultante por ratio de mayor a menor
;; 3. genero un mapa con los índices de los objetos en el array (como claves)
;;     que apuntan a los propios objetos (como valores)
;; Parámetros:
;; - objects: los objetos iniciales definidos en el problema
(defn arrange-objects [objects]

;; Función que devuelve el valor de aptitud o conveniencia de un individuo y
;; el volumen de la mochila que rellena un individuo antes de desbordarla.
;; Procedimiento:
;; - se queda con los objetos que tiene en la mochila (gen true)
;;   ordenados por ratio descendente
;; - acumula los valores de los objetos, mientras quepan en la mochila
;; - devuelve el valor acumulado y el volumen acumulado
;; Parámetros:
;; - individual: el individuo
(defn fitness-and-volume [individual]

;; Función que devuelve el valor de aptitud o conveniencia de un individuo
;; Parámetros:
;; - individual: el individuo
(defn fitness [individual]

;; Función que devuelve el volumen de mochila que rellena un individuo
;; antes de desbordarla.
;; Parámetros:
;; - individual: el individuo
(defn volume [individual]

;; Devuelve la representación de un individuo como el conjunto de objetos
;; que introduce en la mochila.
(defn decode [individual]

;; Genera aleatoriamente un individuo.
(defn rand-individual []

;; Genera aleatoriamente una población.
(defn rand-population []

;; Selecciona el primer elemento de una lista con probabilidad stochastic-prob.
;; De no ser seleccionado, selecciona el primero del resto con probabilidad
;; stochastic-prob también. Y así hasta agotar la lista.
(defn first-stochastic [col]

;; Selecciona por torneo estocástico.
;; - population: población de la que se selecciona
;; - size: número de individuos a seleccionar
(defn tournament-stochastic [population size]
```

```
;; Cruza dos padres por un punto o devuelve los padres tal cual, dependiendo
;; de la probabilidad de cruce.
;; - parent1: primer padre
;; - parent2: segundo padre
(defn crossover-one-point [parent1 parent2]

;; Muta los genes de un individuo atendiendo a una probabilidad de mutación
;; data por 1/número de objetos del individuo.
;; - individual: el individuo a mutar
(defn mutate [individual]

;; Determina si se ha alcanzado el máximo de generaciones del problema.
;; - generation: la generación en curso
(defn too-much-generations? [generation]

;; Determina si se llevan demasiadas generaciones sin que aparezcan individuos
;; con fitness mejorado.
;; - generation: la generación en curso
;; - best: el mejor individuo de la generación
(defn too-much-idle? [generation best]

;; Determina si la evolución ha llegado a su fin, bien por haberse alcanzado
;; demasiadas generaciones, bien por llevar demasiadas generaciones sin que se
;; incremente el fitness del mejor individuo.
;; - generation: la generación en curso
;; - best: el mejor individuo de la generación
(defn done? [generation best]

;; Construye la nueva generación a partir de la generación actual
;; aplicando el modelo generacional.
;; - population: la población actual
(defn build-offspring [population]

;; Inicializa y lleva a cabo la evolución. Reporta el resultado.
;; Devuelve el mejor individuo encontrado.
;; - objects: los objetos a introducir en la mochila.
;; - config: mapa con la configuración del experimento.
;; La configuración adopta esta forma:
;; {:pack-size 500
;;  :rand-gen-prob 1/2
;;  :population-size 10
;;  :stochastic-prob 9/10
;;  :tournament-size 2
;;  :replacement true
;;  :crossover-prob 3/4
;;  :max-generations 100
;;  :idle-generations 5
```

```
;; :report-delta 1
;; :name "Nombre del experimento"}
;; Los objetos son un array con esta forma:
;; [{"nombre 1" valor-1 volumen-1}
;;  {"nombre 2" valor-2 volumen-2}
;;  ...
;; ]
(defn go-live [conf objs]

  ;; Inicializa y lleva a cabo la evolución. Reporta el resultado.
  ;; Devuelve el mejor individuo encontrado.
  ;; - path: ruta al fichero con objetos y configuración a utilizar
  ;; - config-override: parámetros de la configuración indicada en el fichero
  ;;                      que se desean sobrescribir en esta ejecución.
  ;;                      Tiene el mismo formato que config.
  ;; El fichero tiene que tener un formato como el siguiente:
  ;; {:config {:pack-size 500
  ;;           :rand-gen-prob 1/2
  ;;           :population-size 10
  ;;           :stochastic-prob 9/10
  ;;           :tournament-size 2
  ;;           :replacement true
  ;;           :crossover-prob 3/4
  ;;           :max-generations 100
  ;;           :idle-generations 5
  ;;           :report-delta 1
  ;;           :name "amanas: Todo desde fichero 1"}}
  ;; :objects [{"objeto 1" 150 9}
  ;;           [{"objeto 2" 120 8}
  ;;            ...]}
  (defn go-live-from-file [path & [config-override]]

    ;; Función que permite ejecutar el algoritmo invocando el jar ejecutable
    ;; desde una consola.
    ;; El comando para llamar al algoritmo es:
    ;; java -jar ejecutable.jar simple|complex tournament-size
    ;; Por ejemplo, se puede llamar con:
    ;; java -jar ejecutable.jar simple 5
    ;; Esta llamada ejecutará el experimento, cuya evolución puede verse en:
    ;; http://amanas.ml/ce/status.html
    ;; seleccionando en el combobox el experimento con nombre:
    ;; profe: simple - 5
    (defn -main [& [type tour :as args]]
```


5 Formas de ejecutar el algoritmo

Son tres, que describo a continuación.

5.1 Ejecución desde consola

En las indicaciones de la práctica se pide además proporcionar un ejecutable que se pueda lanzar sin necesidad de compilar el código o utilizar un IDE.

En mi caso, todo el proyecto lo compilo en un ejecutable java, **ejecutable.jar**, que adjunto.

La parametrización que permito en este tipo de ejecución del algoritmo es:

- **primer parámetro: el conjunto de datos utilizado, 'simple' o 'complex'**
- **segundo parámetro: el tamaño en la selección por torneo**

Por lo tanto, en un sistema en el que tengamos una máquina virtual java, podremos lanzar un ejecución del algoritmo que utilizase el conjunto de datos complex con tamaño de torneo, por ejemplo 7, haciendo esta simple llamada:

```
java -jar ejecutable.jar complex 7
```

Después, o mientras la ejecución ocurre, podemos visitar <http://amanas.ml/ce/status.html> y seleccionar en el combobox el experimento con nombre **"profe: complex - 7"**. De este modo se puede monitorizar la evolución del aprendizaje llamando al algoritmo desde consola.

5.2 Ejecución programática del algoritmo

Por último, es necesario indicar el modo de representación de los metaparámetros del algoritmo y la forma de iniciar la ejecución del mismo.

En la implementación que entrego **todos los metaparámetros del algoritmo son configurables** en el momento de su lanzamiento.

De este modo, la función que lleve a cabo la ejecución de un experimento es:

```
(defn go-live [conf objs]
  ...
```

a la que se le pasa un mapa con los parámetros de configuración y el conjunto de objetos.

Los parámetros de configuración son los siguientes:

```
{;; El tamaño de la mochila
:pack-size 500 ;; [100,10.000] [10.000,1.000.000]
;; Probabilidad de activación de los genes cuando se genera un individuo
:rand-gen-prob 1/2
;; Tamaño inicial de la población
:population-size 10 ;; [10,100]
;; Probabilidad utilizada para selección estocástica en el torneo
```

```

:stochastic-prob 9/10
;; Número de individuos seleccionados por ronda en selección por torneo
:tournament-size 2 ;; Máxima explotación
;; Torneo con o sin reemplazamiento
:replacement true
;; Probabilidad de mezcla en crossover
:crossover-prob 3/4 ;; [0.6,0.9]
;; Número de generaciones tope para el experimento
:max-generations 100
;; Número de generaciones sin mejora del fitness que se permiten
;; antes de dar por acabado el exp.
:idle-generations 5
;; Cada cuantas generaciones se reporta el estado al web-service
;; en la nube para su visualización.
:report-delta 1}

```

Por ejemplo, una ejecución del algoritmo de aprendizaje podría hacerse así:

```

;; Un conjunto de objetos de prueba
(def some-objects (map #(rand-object % 20 20) (range 200)))

(decode (go-live {:name "amanas: experimento 1"
                  :pack-size 500
                  :rand-gen-prob 1/2
                  :population-size 10
                  :stochastic-prob 9/10
                  :tournament-size 2
                  :replacement true
                  :crossover-prob 3/4
                  :max-generations 100
                  :idle-generations 5
                  :report-delta 1}
                  some-objects))

```

5.3 Ejecución desde fichero del algoritmo

Sin embargo, en las indicaciones sobre como realizar la práctica se hace saber que **sería recomendable que se pudiera proceder a la ejecución de un experimento leyendo tantos los datos como la configuración desde un fichero**. Por lo tanto, también se permite proceder de este modo haciendo una llamada como la siguiente:

```

;; Inicializa y lleva a cabo la evolución. Reporta el resultado.
;; Devuelve el mejor individuo encontrado.
;; - path: ruta al fichero con objetos y configuración a utilizar
;; - config-override: parámetros de la configuración indicada en el fichero
;;                   que se desean sobrescribir en esta ejecución.
;;                   Tiene el mismo formato que config.
;; El fichero tiene que tener un formato como el siguiente:

```

```

;; {:config {:pack-size 500
;;           :rand-gen-prob 1/2
;;           :population-size 10
;;           :stochastic-prob 9/10
;;           :tournament-size 2
;;           :replacement true
;;           :crossover-prob 3/4
;;           :max-generations 100
;;           :idle-generations 5
;;           :report-delta 1
;;           :name "amanas: Todo desde fichero 1"}
;; :objects [{"objeto 1" 150 9]
;;           ["objeto 2" 120 8]
;;           ...]}
(defn go-live-from-file [path & [config-override]]

```

Obsérvese que en este caso, para poder utilizar un mismo fichero con distintas configuraciones, se ofrece la posibilidad de utilizar el parámetro `config-override`, en el que se indicarían los parámetros y valores que queremos sobrescribir sobre aquellos indicados en la configuración del fichero.

Un ejemplo de fichero que cumpliría con los requisitos anteriores sería:

```

{:pack-size 400
 :population-size 5
 :tournament-size 5
 :replacement true
 :rand-gen-prob 5/10
 :stochastic-prob 8/10
 :crossover-prob 5/10
 :max-generations 200
 :idle-generations 20
 :report-delta 1
 :name "amanas: Rosseta Code desde fichero"
 :objects [{"map" 150 9]
           ["compass" 35 13]
           ["water" 200 153]
           ["sandwich" 160 50]
           ["glucose" 60 15]
           ...]}

```

Y un ejemplo de ejecución del algoritmo desde fichero sobrescribiendo parte de la configuración sería:

```

(go-live-from-file "resources/data/simple.edn"
  {:name "amanas: simple - tournament 5 - run 0"
   :tournament-size 5})

```

Y con estas aclaraciones, creo que ya podemos pasar a comentar las experimentaciones que he realizado siguiendo el guión de la práctica.

6 Evaluación Experimental

Tal como se indica en la práctica, he generado dos conjuntos de objetos, uno simple y otro complejo. He ido variando el número de individuos seleccionados en cada ronda de la fase de selección por torneo para así poder contrastar las implicaciones de tal parámetro en el rendimiento de la búsqueda de la solución.

Para cada tamaño de la selección por torneo, he ejecutado el algoritmo varias veces, de modo que pueda obtener medias de los rendimientos y poder así realizar un análisis más consistente.

Todas las ejecuciones que he realizado han quedado guardadas en el servicio que tengo publicado en la nube y que permite revisar tranquilamente tales ejecuciones. De todos modos, para evitar problemas de eliminado de caches, también he guardado los resultados en la carpeta “results” que se adjunta con este documento.

Por ejemplo, la Figura 2 muestra el resultado de la primera ejecución del algoritmo para el conjunto simple con 2 individuos en la selección por torneo.

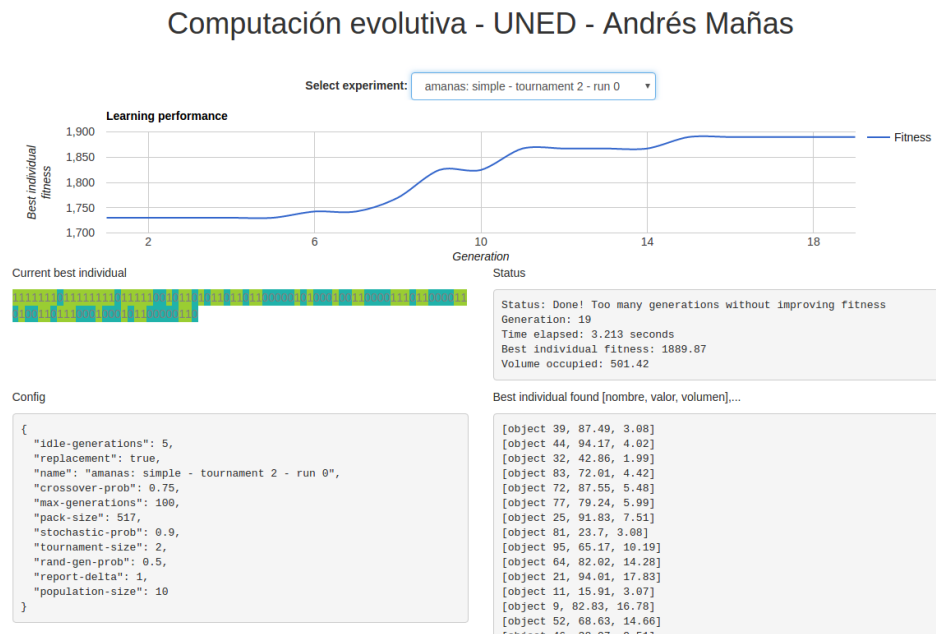


Figura 2. Caso simple con tamaño de torneo 2

6.1 Análisis de resultados

Con el siguiente script podemos agregar todos los resultados y así poder tomar medias. La forma más sencilla es descargarlos desde el servidor y procesarlos aquí. Para prevenir posibles accidentes (que se borre la caché del servidor, guardo una copia de los resultados en el proyecto).

```
In [141]: import os.path
          import json
          import requests
```

```
import pandas as pd

def get_exp_name(type,tournament,run):
    template='amanas: %s - tournament %s - run %s'
    return template % (type,tournament,run)

def get_exp_url(type,tournament,run):
    template='http://amanas.ml/ce/service.php?action=get&name=%s'
    return template % get_exp_name(type,tournament,run)

def get_exp_path(type,tournament,run):
    template='/home/ubuntu/ai/ce/pl/results/%s.json'
    return template % get_exp_name(type,tournament,run)

def get_exp_data(type,tournament,run):
    path=get_exp_path(type,tournament,run)
    if os.path.isfile(path):
        with open(path) as data:
            return json.load(data)
    url=get_exp_url(type,tournament,run)
    data=requests.get(url).json()
    with open(path, 'w') as fp:
        json.dump(data, fp)
    return data

def to_DataFrame(type,tournaments,runs):
    data = []
    for tour in tournaments:
        for run in runs:
            d = get_exp_data(type,tour,run)
            data += [[type,tour,run,d['best-volume'],
                        d['best-fitness'],d['generation'],
                        (d['current-time']-d['start-time'])/1000]]
    df=pd.DataFrame(data)
    df.columns=['type','tournament','run','best-volume',
                'best-fitness','generation','seconds']
    return df
```

6.2 Experimento simple

En el caso del experimento simple, he generado un fichero de datos que se puede consultar en “resources/data/simple.edn”. El experimento se caracteriza por:

- 10 individuos
- 100 objetos
- con valor y volumen aleatoriamente entre [1, 100]
- y con capacidad de la mochila un valor aleatorio en el intervalo real [100, 10.000]

Para cada ejecución del algoritmo, he utilizado estos mismos datos pero he ido cambiando dos propiedades de la configuración (el nombre del experimento y el tamaño de la ronda por torneo).

Así, finalmente he conseguido las siguientes ejecuciones, que se pueden consultar en <http://amanas.ml/ce/status.html>

Los resultados obtenidos son los siguientes:

```
In [142]: simpleDF=to_DataFrame('simple',[2,3,4,5,6,10],range(10))
          print(simpleDF)
```

	type	tournament	run	best-volume	best-fitness	generation	seconds
0	simple	2	0	515.80	1831.38	31	4.949
1	simple	2	1	513.59	1988.19	28	4.351
2	simple	2	2	503.62	1890.99	21	3.295
3	simple	2	3	491.51	1799.69	10	1.799
4	simple	2	4	506.70	1777.14	10	1.726
5	simple	2	5	505.32	1965.96	21	4.579
6	simple	2	6	511.06	1832.96	14	2.347
7	simple	2	7	510.31	1706.32	15	2.505
8	simple	2	8	504.56	1847.42	15	2.550
9	simple	2	9	512.76	1893.51	10	1.729
10	simple	3	0	511.75	1792.64	11	1.853
11	simple	3	1	512.91	1647.21	6	1.127
12	simple	3	2	516.46	1710.22	10	1.645
13	simple	3	3	500.35	1721.76	6	1.035
14	simple	3	4	506.82	1893.23	16	2.682
15	simple	3	5	496.01	1918.09	29	4.715
16	simple	3	6	496.35	1784.91	7	1.197
17	simple	3	7	510.28	1839.43	17	2.837
18	simple	3	8	507.10	1876.43	11	1.868
19	simple	3	9	515.51	1726.58	13	2.102
20	simple	4	0	509.12	1811.24	13	2.239
21	simple	4	1	488.67	1839.10	13	2.142
22	simple	4	2	514.90	1914.26	13	2.329
23	simple	4	3	515.89	1668.33	6	2.057
24	simple	4	4	506.84	1877.97	16	2.626
25	simple	4	5	514.38	1888.85	19	3.173
26	simple	4	6	515.90	1877.46	21	3.364
27	simple	4	7	515.81	1835.59	11	1.837
28	simple	4	8	512.25	1915.86	15	2.492
29	simple	4	9	510.33	2036.28	21	3.607
30	simple	5	0	515.49	1794.37	17	2.766
31	simple	5	1	496.79	1894.97	17	3.914
32	simple	5	2	515.90	1885.75	13	2.209
33	simple	5	3	507.26	1672.82	9	1.566
34	simple	5	4	510.69	2034.27	20	3.343
35	simple	5	5	512.40	1994.26	25	4.174
36	simple	5	6	499.16	1852.76	24	4.594

37	simple	5	7	508.92	1782.60	10	1.759
38	simple	5	8	501.27	1719.95	13	2.222
39	simple	5	9	516.25	1860.77	11	1.842
40	simple	6	0	498.96	1633.97	7	1.214
41	simple	6	1	505.90	1763.76	7	1.215
42	simple	6	2	512.05	1945.27	20	3.218
43	simple	6	3	513.00	1801.96	10	1.653
44	simple	6	4	499.04	1791.12	15	2.561
45	simple	6	5	514.42	1909.29	19	3.201
46	simple	6	6	516.50	1752.97	9	1.555
47	simple	6	7	511.44	1879.01	9	1.586
48	simple	6	8	513.12	1898.53	10	1.715
49	simple	6	9	511.73	1882.00	14	2.332
50	simple	10	0	511.07	1794.53	13	2.206
51	simple	10	1	509.41	2018.32	19	3.228
52	simple	10	2	516.15	1746.17	6	1.109
53	simple	10	3	512.07	1902.08	13	2.230
54	simple	10	4	494.86	1851.11	7	1.284
55	simple	10	5	516.01	1854.98	12	2.121
56	simple	10	6	516.17	1900.01	14	2.522
57	simple	10	7	515.87	1821.77	9	1.524
58	simple	10	8	516.82	1998.26	17	2.953
59	simple	10	9	502.06	1941.80	24	4.075

De dónde, agrupando por el tamaño del torneo:

```
In [143]: grouped=simpleDF.groupby('tournament')
```

podemos observar los valores medios agrupados por tamaño del torneo:

```
In [144]: print(grouped.mean())
```

	run	best-volume	best-fitness	generation	seconds
tournament					
2	4.5	507.523	1853.356	17.5	2.9830
3	4.5	507.354	1791.050	12.6	2.1061
4	4.5	510.409	1866.494	14.8	2.5866
5	4.5	508.413	1849.252	15.9	2.8389
6	4.5	509.616	1825.788	12.0	2.0250
10	4.5	511.049	1882.903	13.4	2.3252

y las desviaciones medias agrupadas por tamaño del torneo:

```
In [145]: print(grouped.std())
```

	run	best-volume	best-fitness	generation	seconds
tournament					
2	3.02765	6.983955	85.069391	7.531416	1.236304
3	3.02765	7.508329	89.710293	6.915361	1.102420
4	3.02765	8.270967	93.288730	4.685676	0.598810

5	3.02765	7.164415	112.609439	5.685264	1.097257
6	3.02765	6.210461	94.425764	4.737557	0.756431
10	3.02765	7.303104	86.799704	5.521674	0.917717

6.2.1 Curvas de evolución

En las figuras 3 y 4 vemos un par de ejecuciones (torneo tamaño 2 y torneo tamaño 10).

Computación evolutiva - UNED - Andrés Mañas

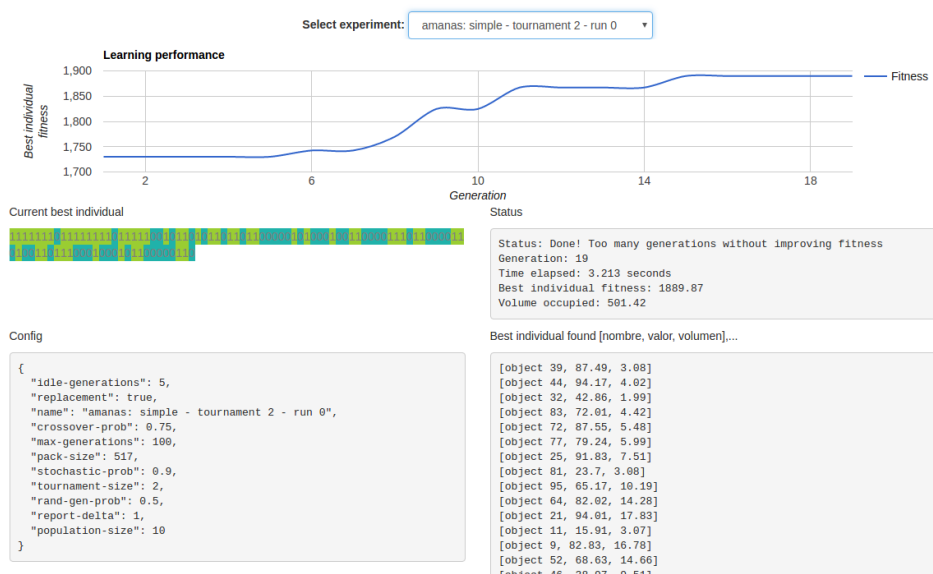


Figura 3. Caso simple con tamaño de torneo 2

6.2.2 Conclusiones del experimento simple

En el caso simple observo que **el tamaño del torneo parece condicionar el rendimiento del algoritmo**.

El tiempo que tarda cada ejecución parece mantenerse estable en torno a los 2.5 segundos y la desviación en torno al segundo.

El número de generaciones necesarias parece disminuir según el torneo se hace con mayor número de individuos (nótese que hay que restar 5 generaciones de las que se muestran necesarias, porque es el umbral que tengo configurado para acabar la ejecución si no se mejora el fitness durante esas generaciones). Por eso, parece que con tamaño de torneo superior a 5 son necesarias sólo 8 generaciones para dar con la solución “óptima” (habida cuenta que en realidad no sabemos cuál es - no sé si se llega al máximo global - intuyo que sí).

Observo además que a mayor tamaño de torneo, mayor fitness se alcanza (el mayor fitness se alcanza con torneo tamaño 10).

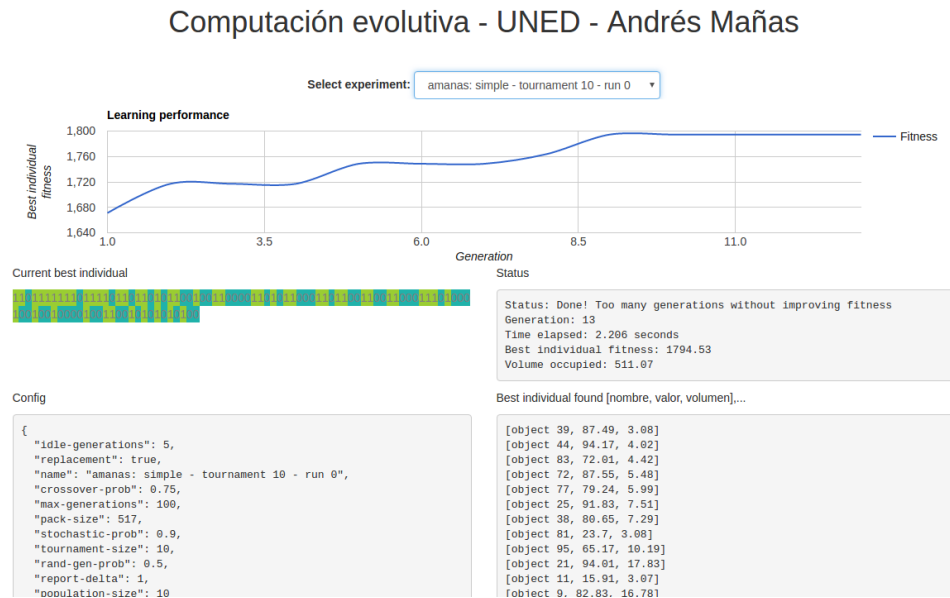


Figura 4. Caso simple con tamaño de torneo 10

Por lo tanto, al amparo de los datos experimentales arrojados por mis pruebas, me aventuro a concluir que en experimentos pequeños tanto la **explotación** (tamaño de torneo pequeño) como la **exploración** (tamaño de torneo mayor) conducen correctamente a la solución.

Pero, en problemas pequeños, parece que mayor exploración hace que la solución se encuentre con menos generaciones

Es decir, evolucionar utilizando la diversidad genética de individuos con inferior fitness parece agilizar el proceso de búsqueda de la solución en problemas de complejidad pequeña.

6.3 Experimento complejo

En el caso del experimento complejo, he generado un fichero de datos que se puede consultar en “resources/data/complex.edn”. El experimento se caracteriza por:

- 100 individuos
- 1.000 objetos
- con valor y volumen aleatoriamente entre [1, 100]
- y con capacidad de la mochila un valor aleatorio en el intervalo real [10.000, 1.000.000]

Para cada ejecución del algoritmo, he utilizado estos mismos datos pero he ido cambiando dos propiedades de la configuración (el nombre del experimento y el tamaño de la ronda por torneo).

Así, finalmente he conseguido las siguientes ejecuciones, que se pueden consultar en <http://amanas.ml/ce/status.html>

Desgraciadamente, Clojure no es un lenguaje que destaque en rendimiento de computación, por lo que he tenido que disminuir un poco la dimensión del experimento complejo propuesto en la práctica. Por eso no puedo utilizar 10.000 objetos sino 1.000 y el tamaño de la mochila

lo disminuye apropiadamente también. En otro caso, con el ordenador que tengo no creo que hubiera podido hacer los experimentos que presento a continuación.

Los resultados obtenidos son los siguientes:

```
In [146]: complexDF=to_DataFrame('complex',[2,3,4,5,6,10],range(5))
          print(complexDF)
```

	type	tournament	run	best-volume	best-fitness	generation	seconds
0	complex	2	0	15150.44	27840.15	100	179.070
1	complex	2	1	15223.73	26567.51	42	73.577
2	complex	2	2	15199.36	27395.51	71	127.434
3	complex	2	3	15210.09	27981.86	100	178.581
4	complex	2	4	15223.42	26720.83	61	111.899
5	complex	3	0	15202.33	28680.32	100	186.781
6	complex	3	1	15227.86	28789.40	100	190.147
7	complex	3	2	15236.04	28214.49	100	192.701
8	complex	3	3	15227.74	28394.13	88	168.250
9	complex	3	4	15211.64	28649.67	100	194.109
10	complex	4	0	15230.64	29542.26	100	201.274
11	complex	4	1	15221.38	29252.36	100	198.896
12	complex	4	2	15219.48	29387.91	100	205.197
13	complex	4	3	15195.86	29307.62	100	199.350
14	complex	4	4	15198.31	29122.33	100	199.423
15	complex	5	0	15236.31	29732.72	100	211.799
16	complex	5	1	15226.39	29732.89	100	215.348
17	complex	5	2	15226.68	29647.90	100	211.344
18	complex	5	3	15213.74	29627.18	100	208.749
19	complex	5	4	15236.49	29410.93	100	208.806
20	complex	6	0	15227.74	29527.41	100	210.258
21	complex	6	1	15225.28	29743.44	100	206.033
22	complex	6	2	15236.73	29552.85	100	204.719
23	complex	6	3	15236.35	29669.85	100	206.211
24	complex	6	4	15222.84	29476.11	100	203.478
25	complex	10	0	15214.65	30041.17	100	236.423
26	complex	10	1	15229.73	30071.88	100	234.273
27	complex	10	2	15223.72	29992.23	100	241.219
28	complex	10	3	15236.96	29908.14	100	236.123
29	complex	10	4	15226.51	29879.04	100	244.519

De dónde, agrupando por el tamaño del torneo:

```
In [147]: grouped=complexDF.groupby('tournament')
```

podemos observar los valores medios agrupados por tamaño del torneo:

```
In [148]: print(grouped.mean())
```

	run	best-volume	best-fitness	generation	seconds
tournament					
2	2.0	15201.408	27301.172	74.8	134.1122

3	2.0	15221.122	28545.602	97.6	186.3976
4	2.0	15213.134	29322.496	100.0	200.8280
5	2.0	15227.922	29630.324	100.0	211.2092
6	2.0	15229.788	29593.932	100.0	206.1398
10	2.0	15226.314	29978.492	100.0	238.5114

y las desviaciones medias agrupadas por tamaño del torneo:

```
In [149]: print(grouped.std())
```

	run	best-volume	best-fitness	generation	seconds
tournament					
2	1.581139	30.249414	639.880822	25.252723	45.280274
3	1.581139	13.736747	235.050780	5.366563	10.520681
4	1.581139	15.271463	156.350558	0.000000	2.606636
5	1.581139	9.337932	131.773720	0.000000	2.707566
6	1.581139	6.403950	109.671231	0.000000	2.553424
10	1.581139	8.182642	83.184564	0.000000	4.226472

6.3.1 Curvas de evolución

En las figuras 5 y 6 vemos un par de ejecuciones (torneo tamaño 2 y torneo tamaño 10).

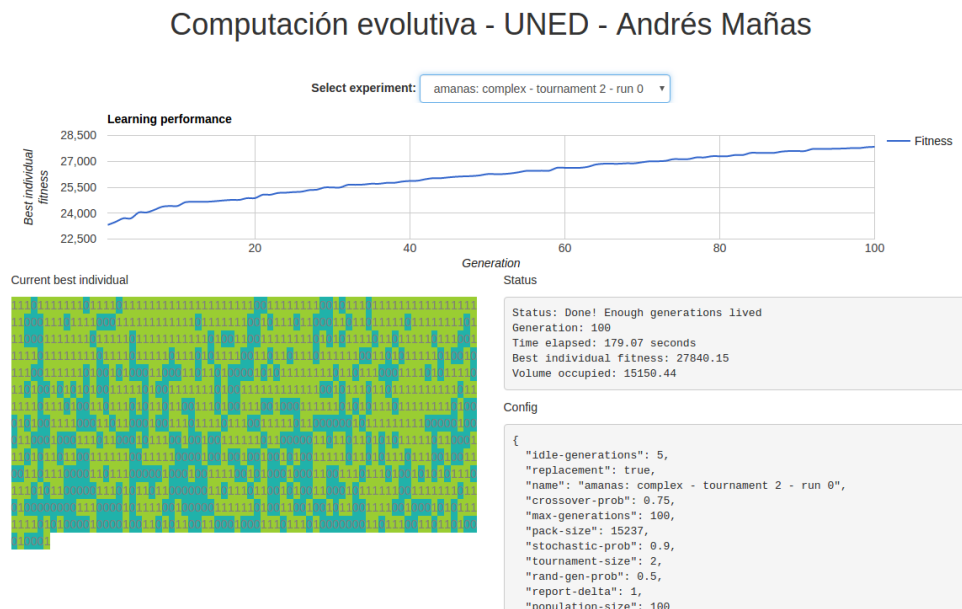


Figura 5. Caso complejo con tamaño de torneo 2

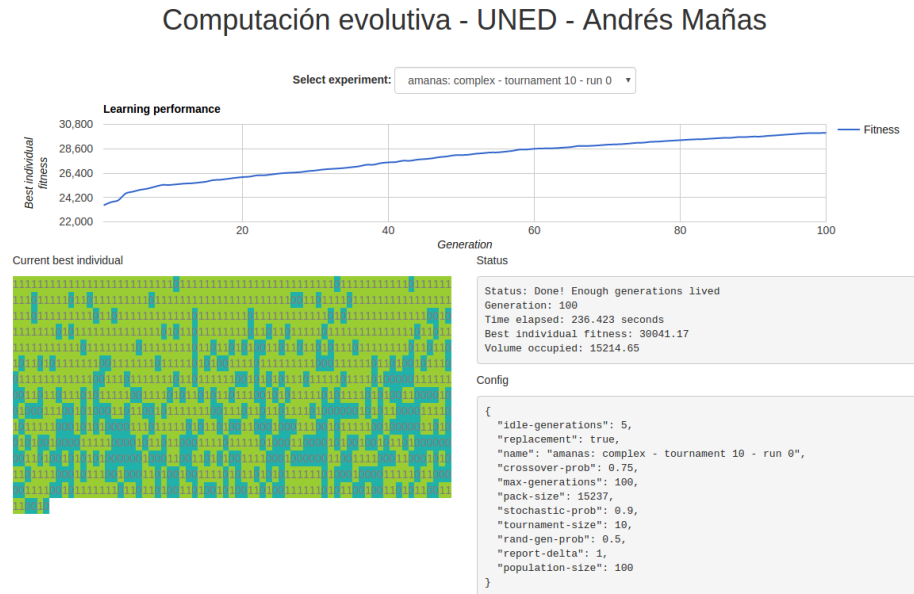


Figura 6. Caso complejo con tamaño de torneo 10

6.3.2 Conclusiones del experimento complejo

En el caso simple observo que **el tamaño del torneo parece estar relacionado con la calidad de la solución pero incide notablemente en el tiempo empleado para encontrarla**. A mayor tamaño de torneo, mayor fitness del mejor individuo encontrado.

El tiempo que tarda cada ejecución crece considerablemente según se aumenta el tamaño del torneo (más individuos de mayor complejidad a los que calcularles el fitness cuando participan en el torneo).

El número de generaciones necesarias aumenta según el torneo se hace con mayor número de individuos. Es decir, cuando el tamaño del torneo es mayor, el fitness sigue creciendo durante más tiempo y se llega a valores superiores a los obtenidos con torneos pequeños.

Al amparo de los datos experimentales arrojados por mis pruebas, concluyo que en experimentos complejos la **exploración** (tamaño de torneo mayor) parece conducir a resultados mejores (individuos con mayor fitness). Sin embargo, el coste computacional es muy elevado.

Observando los gráficos de aprendizaje (figuras 5 y 6), no parece que se produzca sobreentrenamiento puesto que el crecimiento se mantiene bastante constante.

Es decir, evolucionar utilizando la diversidad genética de individuos con inferior fitness ralentiza el proceso de búsqueda de la solución en problemas complejos pero conduce a mejores soluciones.