Universidad Autónoma Metropolitana Unidad
Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Adaptación de una técnica heurística para resolver un problema de programación de horarios

Josue Daniel Castillo Cruz
207302447

Trimestre 2013 Otoño

Asesores:

Roman Anselmo Mora Gutiérrez
Profesor Asociado
Departamento de Sistemas

Eric Alfredo Rincón García
Profesor Asociado
Departamento de Sistemas

TABLA DE CONTENIDO

Índice general

Parte I

REPORTE FINAL

1

RESUMEN

De acuerdo al plan de estudios vigente en la Universidad Autónoma Metropolitana (UAM), se requiere que el alumno cumpla con un mínimo de créditos para poder concluir su carrera. Los alumnos acumulan créditos inscribiendo y cursando diversas Unidades de Enseñanza y Aprendizaje (UEA) las cuales, de ser aprobadas, aportarán una cantidad determinada de créditos al alumno. La carrera se divide en unidades lectivas[1] llamadas trimestres los cuales tienen un límite máximo de créditos, por lo que todas las UEA del plan de estudios deberán ser distribuidas a lo largo de estos trimestres.

El problema descrito en el párrafo anterior es un "Problema de Empaquetamiento con Restricciones de Precedencia" (*BPP-P*[**?**] por sus siglas en inglés) en el que dado un conjunto de contenedores con una capacidad preestablecida, un conjunto de elementos ponderados y un con-

1 Periodo de tiempo destinado para dar lección en los establecimientos de enseñanza.

junto de reglas de precedencia entre estos elementos, estamos interesados en determinar el número mínimo de contenedores necesarios para acomodar todos los elementos de tal forma que estos cumplan todas las reglas de precedencia.

En este proyecto terminal se adaptó la técnica heurística "Optimización por Enjambre de Partículas" (*PSO*[**?**] por sus siglas en inglés) para determinar el número de trimestres requeridos para que un estudiante de la carrera de Ingeniería Física de la Universidad Autónoma Metropolitana complete su licenciatura, para lo cual se creó un banco de instancias académicas a partir del plan de estudios antes mencionado, que simulan la selección de UEA que podrían realizar un conjunto de estudiantes.

El uso de la técnica heurística *PSO* en estas instancias fue útil para determinar su eficiencia en este tipo de problemas en comparación con otros métodos del estado del arte. Además, esta técnica permitió obtener diversas soluciones de buena calidad que ayudaron a determinar que tan rígido es el plan de estudios.

2

OBJETIVOS

2.1 OBJETIVO GENERAL

Adaptar la técnica heurística "Optimización por Enjambre de Partículas" e implementar un algoritmo basado en ella, para resolver un problema de programación de horarios.

2.2 OBJETIVOS ESPECÍFICOS

1. Adaptar la técnica heurística "Optimización por Enjambre de Partículas" para trabajar en un espacio de búsqueda discreto.

2. Implementar un algoritmo basado en la adaptación realizada.

3. Calibrar el algoritmo sobre un conjunto reducido de instancias de prueba.

4. Resolver las instancias de prueba para probar el algoritmo propuesto.

5. Analizar los resultados y comparar el desempeño de la técnica propuesta con el de algoritmos del estado del arte.

3

INTRODUCCIÓN

3.1 PROBLEMA DE EMPAQUETAMIENTO

Dados n elementos donde cada uno tiene asociado un peso no negativo $\omega_j (j = 1, 2, \ldots, n)$ y m contenedores de capacidad idéntica C, el "Problema de Empaquetamiento" (*BPP*[**?**] por sus siglas en inglés) busca el número mínimo de contenedores necesarios para acomodar todos los elementos. Dado un conjunto de precedencias entre los elementos, el BPP-P es una generalización del BPP que requiere el ordenamiento de los contenedores de tal forma que todas las reglas de precedencia sean satisfechas. Intuitivamente hablando, la precedencia impone que los sucesores de un elemento deberán estar en contenedores que siguen al contenedor que empaca a este elemento.

Formalmente, consideremos un grafo simple acíclico $G_0 = (V_0, A_0)$ donde $V_0 = 1, 2, \ldots, n$ contiene un vértice j para cada elemento j de un BPP, y donde cada arco

$(j,k) \in A_0$ representa una relación de precedencia entre el elemento j y el elemento k. En una solución factible al BPP-P, el índice del contenedor que almacena a j, deberá ser estrictamente menor al del contenedor que almacena al elemento k, para todo $(j,k) \in A_0$. El BPP-P es NP-duro[**?**] en el sentido que generaliza al BPP.

3.2 OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS

La Optimización por Enjambre de Partículas es una técnica heurística de búsqueda basada en la simulación del comportamiento social observado en las aves dentro de una parvada.

En *PSO*, cada individuo representa una solución que toma la forma de una partícula que viaja a través de un espacio de búsqueda de varias dimensiones. El cambio en la posición de las partículas dentro del espacio de búsqueda está basado en la tendencia social de cada individuo al emular el éxito de otros individuos. Esto se consigue haciendo que cada partícula del enjambre recuerde su mejor posición obtenida y la mejor posición obtenida por el enjambre completo, donde la calidad de cada solución es calculada evaluando su posición con una función objetivo.

En la versión estándar del algoritmo PSO, la *i-ésima* partícula del enjambre puede ser representada por un vector de posición n-dimensional $X_i = (x_{i1}, x_{i2}, ..., x_{in})$ y su ve-

locidad como $V_i = (v_{i1}, v_{i2}, ..., v_{in})$. Además, se considera la mejor posición visitada por la partícula como $P_{iLbest} = (p_{i1}, p_{i2}, ..., p_{in})$ y la mejor posición explorada por el enjambre hasta el momento como $P_{Gbest} = (p_{g1}, p_{g2}, ..., p_{gn})$. Así, la posición de la partícula y su velocidad son actualizadas en cada iteración usando las siguientes ecuaciones:

$$V_i(t+1) = \omega V_i(t) + \alpha r_1(P_{Gbest} - P_i) + \beta r_2(P_{iLbest} - P_i) \quad (1)$$

$$P_i(t+1) = P_i(t) + V_i(t+1) \quad (2)$$

Donde $V_i(t), V_i(t+1)$ son las velocidades de la partícula i al momento de la iteración t y $t+1$; P_{iLbest} es la mejor posición conocida por la partícula i, y P_{Gbest} es la mejor posición conocida por todas las partículas hasta la iteración t; $P_i(t)$, $P_i(t+1)$ son las posiciones de la partícula i al momento de la iteración t y $t+1$; α y β son constantes positivas que determinan la tendencia de la partícula de seguir a la mejor posición/solución global y personal respectivamente; r_1, r_2 son variables aleatorias con una distribución uniforme entre 0 y 1; y ω es la inercia que muestra el efecto del vector velocidad previo en el nuevo vector velocidad.

El algoritmo 1 provee el pseudocódigo correspondiente.

Algoritmo 1: Algoritmo PSO

1 Inicializar la población de k partículas con posiciones y velocidades aleatorias.
2 Evaluar cada partícula de acuerdo a la función objetivo.
3 Actualizar la mejor posición global y personal.
4 **while** *Los criterios de paro no se cumplan* **do**
5 Para cada partícula k, actualiza la velocidad y posición usando las ecuaciones 1 y 2.
6 Evaluar cada partícula de acuerdo a la función objetivo.
7 Actualizar la mejor posición global y personal.
8 Memorizar la mejor posición encontrada hasta el momento.
9 **end**

4

DESARROLLO

4.1 DESCRIPCIÓN DEL PROBLEMA

De acuerdo al plan de estudios vigente en la Universidad Autónoma Metropolitana , se requiere que el alumno cumpla con un mínimo de créditos para poder concluir su carrera. Los alumnos acumulan créditos inscribiendo y cursando diversas Unidades de Enseñanza y Aprendizaje (UEA) las cuales, de ser aprobadas, aportarán una cantidad determinada de créditos al alumno. La carrera se divide en unidades lectivas llamadas trimestres los cuales tienen un límite máximo de créditos, por lo que todas las UEA del plan de estudios deberán ser distribuidas a lo largo de estos trimestres.

Las UEA disponibles se pueden clasificar por el tipo de restricciones que deben cumplirse para poder ser inscritas/cursadas.

1. UEA con seriación: requiere haber cursado determinadas UEA para poder ser inscrita.

2. UEA con requisitos de créditos: exige el haber acumulado una cierta cantidad de créditos para poder ser inscrita.

3. UEA sin requisitos: UEA que no requiere haber cursado alguna UEA ni haber acumulado una cierta cantidad de créditos.[1]

Además de las anteriores, se agregaron las siguientes Restricciones adicionales:

1. Límites inferiores y superiores. Se calculó el trimestre mínimo y máximo en el cual una UEA puede existir considerando su seriación y se impidio su asignación fuera de este rango.

4.2 DETALLES DEL PROGRAMA

El algoritmo se implementó utilizando el lenguaje de programación Python[2] en su versión 2.7 y fue ejecutado en el intérprete para python PyPy[3]

El programa lee los datos de un archivo de texto para después crear en memoria un arreglo asociativo donde

1 Una UEA puede pertenecer a más de una de estas clasificaciones.

2 `www.python.org`

3 `www.pypy.org`

cada una de sus entradas contiene una representración de cada UEA. Esta representación almacena los atributos asociados a cada UEA como, clave, nombre, créditos asociados, UEA requeridas y créditos requeridos.

También crea una representación para cada partícula donde cada una tiene asociada un vector posición, un vector velocidad y su calidad.

El programa crea un arreglo de partículas cuyas posiciones y velocidades son inicializadas de forma aleatoria, enseguida, se entra al ciclo principal que evaluará cada partícula, calculará la mejor posición conocida por cada partícula y la mejor posición conocida por todas las partículas para después realizar la actualización de la velocidad y posición utilizando 1 y 2. Este proceso se repite un determinado número de veces o hasta que el programa regrese resultados satisfactorios.

Cabe destacar que *PSO* fue diseñado teniendo en mente un espacio de soluciones continuo y que se han hecho varios intentos por adaptarlo a espacios de búsqueda discreto[**?**] En este proyecto *PSO* se discretizó mediante un redondeo en el cálculo de la velocdad.

Para evitar una convergencia rápida se aplicó un método de exterminación, donde cada cierto número de iteraciones se elimina un 10 % de las partículas y se sustituyen con nuevos individuos cuyos atributos son generados aleatoriamente.

Los resultados son entregados a través de la consola donde se imprimirá el número de corrida. Cada 100 ite-

raciónes se imprimirá la calidad de la mejor partícula conocida por el enjambre (Global) hasta que se encuentre la mejor solución o hasta que se cumplan 2000 iteraciones. Al final de cada corrida se imprimirán los mejores resultados encontrados. El programa se detendrá después de haber cumplido 30 corridas.

4.3 FUNCIÓN OBJETIVO

Para el cálculo de calidad de cada partícula, se evalúa el vector posición asociado a esta, de tal forma que la calidad inicial de la partícula es cero[4] e irá en ascenso de acuerdo a las siguientes reglas.

1. Restricciones de seriación. Si una UEA en la solución viola alguna regla de seriación, se calcula la diferencia de trimestres de estas dos UEA y se le suma esta diferencia a la calidad de la partícula.

2. Exceso de créditos por trimestre. Si un trimestre tiene asociado un número de UEA tal que la suma de los créditos de estas UEA excede el límite establecido por el trimestre, se calcula la diferencia entre este límite y la suma de créditos de las UEA asociadas. Esta diferencia luego es sumada a la calidad de la partícula.

4 Una calidad con un valor absoluto bajo es mejor.

3. Restricciones de créditos requeridos. Si una UEA requiere cierta cantidad de créditos, se calcula la suma de los créditos acumulados hasta un trimestre anterior a esta UEA. Si los créditos acumulados son menores a los créditos requeridos, se realiza la diferencia entre estas dos cantidades y se suma a la calidad de la partícula.

4. Número de trimestres. Para influenciar al algoritmo para que busque soluciones con pocos trimestres, se aplica una penalización que depende del número de trimestres máximos en la solución actual. Esta penalización se suma a la calidad global de la partícula.

4.4 ELECCIÓN DE PARÁMETROS

La técnica *PSO* descrita en secciones anteriores tiene un pequeño número de parámetros que necesitan ser fijados, para esto, se utilizó el siguiente método de constricción[**?**]:

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{3}$$

donde $\phi = \alpha + \beta$

Con esto, $\omega \leftrightarrow \chi$ y $\phi \leftrightarrow \chi\phi$. Así, la configuración óptima corresponde a $\omega = 0.7298$ y $\alpha = \beta = 1.49618$ en 1.

5

PRUEBAS

Las pruebas aquí descritas se realizaron en una computadora con las siguientes características:

1. Procesador: Intel Core i5
2. Memoria Ram: 4 GB DDR3
3. Sistema Operativo: Windows 7

5.1 CALIBRACIÓN DEL ALGORITMO

Para determinar el número de iteraciones y el número de partículas necesarias, se empleó un método de fuerza bruta. Dicho procedimientro consiste en establecer un periodo t, de manera inicial se tiene una configuración arbitraria de los parámetros del algoritmo, denotada como θ, la cual es utilizada para ejecutar el algoritmo, evaluando los resultados obtenidos; Posteriormente, se altera uno de

los parámetros de la configuración θ dando como resultado una configuración θ^1, dicha configuración es ocupada para la ejecución del algoritmo y se evalúan los resultados. Una vez evaluadas ambas configuraciones, se selecciona a aquella configuración que genere los mejores resultados. Varios autores explican que el procedimiento de fuerza bruta es un proceso iterado de prueba y error.

5.2 PRUEBAS REALIZADAS

Se seleccionaron cuatro instancias del plan de estudios de Ingeniería Física[1] y una del plan de estudios de Ingeniería en Computación[2], para las cuales se ejecutaron 30 corridas utilizando el método *PSO* descrito anteriormente, y el "Método de Composición Musical" (*MMC*[**?**] por sus siglas en inglés) realizando 100000 llamadas a la función objetivo y determinando para cada corrida, el tiempo de ejecución y la calidad de la solución entregada[3]. Posteriormente los resultados[4] generados por ambas técnicas heurísticas fueron sujetas a la prueba de Wilkcoxon[**?**].

Cabde destacar que en un principio para las pruebas realizadas con la instancia de Ingeniería en Computación, se utilizaron 50 partículas y solo 10000 llamadas a la fun-

1 Ver Apéndice A.
2 Ver Apéndice A.
3 Número de trimestres y número de restricciones violadas en la solución.
4 Ver Apéndice B.

ción objetivo consiguiendo tiempos menores a un segundo, pero debido al aumento en la dificultad de las instancias restantes, se decidió utilizar 500 partículas para todas las pruebas.

6

ANÁLISIS DE RESULTADOS

6.1 OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS

En la tabla 1 se puede observar que para la instancia de Computación, hay una probabilidad de cero de generar soluciones infactibles. En contraste, en la instancia de Instrumentación I existe una probabilidad 0.3666 de obtener una solución infactible. Tanto el radio superior como el radio inferior fueron calculados usando la prueba de bootstrap con un 95 % de certeza.

En la tabla 2 se muestra el valor promedio, máximo, mínimo y varianza del número de Restricciones violadas para las 30 corridas realizadas sobre cada una de las instancias. Se puede observar con claridad que en la instancia de Computación, se llega a solución factible en todos los casos, lo cual contrasta con la instancia de Instrumentación I donde en promedio se violan 0.6 restricciones en

Instancia	Radio	R Inferior	R Superior
Computación	0	0	0
Instrumentación I	0.3666	0.2	0.5333
Instrumentación II	0.2	0.0666	0.3666
Tecnología I	0.1	0	0.2
Tecnología II	0.0666	0	0.1666

Tabla 1.: Infactibilidad con PSO

cada una de las corridas, lo cual es consistente con lo obtenido en la tabla anterior donde se muestra la probabilidad de obtener soluciones infactibles.

Instancia	$\overline{x}$	Max	Min	σ^2
Computación	0	0	0	0
Instrumentación I	0.6	2	0	0.7310
Instrimentación II	0.3	2	0	0.4241
Tecnología I	0.1	1	0	0.0931
Tecnología II	0.0666	1	0	0.0643

Tabla 2.: Restricciones violadas con PSO

Tanto el límite superior como el límite inferior mostrados en la tabla 3 se obtuvieron del promedio de restricciones violadas y fueron calculados usando la prueba de bootstrap con un 95 % de certeza.

Instancia	Límite Inferior	Límite Superior
Computación	0	0
Instrumentación I	0.3	0.9
Instrumentación II	0.1	0.5333
Tecnología I	0	0.2333
Tecnología II	0	0.1666

Tabla 3.: Límites calculados con prueba Bootstrap sobre número de restricciónes violadas con PSO

En la tabla 4 se muestra un análisis a través de Estadística Descriptiva sobre la Función Objetivo considerando las soluciones factibles e infactibles.

Instancia	$\overline{x}$ F.O.	Máx	Mín	σ^2	σ
Computación	11	11	11	0	0
Instrumentación I	14.1	19	13	2.1620	1.4703
Instrumentación II	13.7333	18	13	1.0575	1.0482
Tecnología I	13.5	15	13	0.3275	0.5723
Tecnología II	14.3	17	13	0.6310	0.7943

Tabla 4.: Soluciones factibles e infactibles con PSO

En la tabla 5 se muestra un análisis a través de Estadística Descriptiva sobre la Función Objetivo considerando solamente las soluciones factibles. Podemos apreciar que los resultados con mejor calidad están dados en la instan-

cia de Computación, mientras que los de peor calidad se entregan la instancia de Tecnología II.

Instancia	$\overline{x}$	Máx	Mín	σ^2	σ
Computación	11	11	11	0	0
Instrumentación I	13.4210	16	13	0.5906	0.7685
Instrumentación II	13.375	14	13	0.2445	0.4945
Tecnología I	13.4074	14	13	0.2507	0.5007
Tecnología II	14.2142	17	13	0.5449	0.7382

Tabla 5.: Soluciones factibles con PSO

En la tabla 6 se muestra un análisis a través de Estadística Descriptiva sobre el tiempo de ejecución. Podemos ver que para la instancia de Computación se requieren en promedio 16 segundos, en contraste, la instancia Tecnología II requiere aproximadamente 5 veces más.

Instancia	$\overline{x}$ t.	Max	Mín	σ^2	σ
Computación	15.69	79.32	3.02	279.96	16.73
Instrumentación I	75.74	130.03	9.83	1883.93	43.40
Instrimentación II	70.51	125.43	16.08	1489.91	38.59
Tecnología I	75.78	131.84	15.05	1355.48	36.81
Tecnología II	80.25	136.70	27.35	1054.94	32.47

Tabla 6.: Tiempo de ejecución

6.2 MÉTODO DE COMPOSICIÓN MUSICAL

En la tabla 1 se puede observar que para la instancia de Computación, hay una probabilidad de cero de generar soluciones infactibles. En contraste, en la instancia de Tecnología I existe una probabilidad 0.3666 de obtener una solución infactible. Tanto el radio superior como el radio inferior fueron calculados usando la prueba de bootstrap con un 95 % de certeza.

Instancia	Radio	R Inferior	R Superior
Computación	0	0	0
Instrumentación I	0.0666	0	0.1666
Instrumentación II	0.0666	0	0.1666
Tecnología I	0.1	0	0.2333
Tecnología II	0.3666	0.2	0.5333

Tabla 7.: Infactibilidad con MMC

En la tabla 8 se muestra el valor promedio, máximo, mínimo y varianza del número de Restricciones violadas para las 30 corridas realizadas sobre cada una de las instancias. Se puede observar con claridad que en la instancia de Computación, se llega a solución factible en todos los casos, lo cual contrasta con la instancia de Tecnología II donde en promedio se violan 0.4333 restricciones en cada una de las corridas, lo cual es consistente con lo obtenido

en la tabla anterior donde se muestra la probabilidad de obtener soluciones infactibles.

Instancia	$\overline{x}$	Max	Min	σ^2
Computación	0	0	0	0
Instrumentación I	0.6666	1	0	0.0643
Instrimentación II	0.6666	2	0	0.0643
Tecnología I	0.1	1	0	0.0931
Tecnología II	0.4333	2	0	0.3919

Tabla 8.: Número de restricciones violadas con MMC

Tanto el límite superior como el límite inferior mostrados en la tabla 9 se obtuvieron del promedio de restricciones violadas y fueron calculados usando la prueba de bootstrap con un 95 % de certeza.

Instancia	Límite inferior	Límite Superior
Computación	0	0
Instrumentación I	0	0.1666
Instrumentación II	0	0.1666
Tecnología I	0	0.2333
Tecnología II	0.2333	0.6666

Tabla 9.: Límites calculados con prueba Bootstrap sobre número de restricciónes violadas con MMC

En la tabla 11 se muestra un análisis a través de Estadística Descriptiva sobre la Función Objetivo considerando las soluciones factibles e infactibles.

Instancia	$\overline{x}$ F.O.	Máx	Mín	σ^2	σ
Computación	11	11	11	0	0
Instrumentación I	13.0666	14	13	0.0643	0.2537
Instrumentación II	13.0666	14	13	0.0643	0.2537
Tecnología I	13.1	14	13	0.0931	0.3051
Tecnología II	13.4333	15	13	0.3919	0.6260

Tabla 10.: Soluciones factibles e infactibles con MMC

En la tabla 10 se muestra un análisis a través de Estadística Descriptiva sobre la Función Objetivo considerando solamente las soluciones factibles.

Instancia	$\overline{x}$	Máx	Mín	σ^2	σ
Computación	11	11	11	0	0
Instrumentación I	13	13	13	0	0
Instrumentación II	13	13	13	0	0
Tecnología I	13	13	13	0	0
Tecnología II	13	13	13	0	0

Tabla 11.: Soluciones factibles con MMC

En la tabla 12 se muestra un análisis a través de Estadística Descriptiva sobre el tiempo de ejecución. Podemos ver que para la instancia de Computación se requieren en promedio 5 segundos, en contraste, la instancia Tecnología II requiere aproximadamente 2 veces más.

Instancia	$\overline{x}$ t.	σ^2	σ
Caomputación	5.5084	16.5353	4.0663
Instrumentación I	17.8825	93.8684	9.6885
Instrimentación II	19.6902	81.5313	9.0294
Tecnología I	22.6902	105.4854	10.2706
Tecnología II	28.6196	138.9308	11.7868

Tabla 12.: Tiempo de ejecución con MMC

Se realizaron dos pruebas de Wilkcoxon, una sobre el número de soluciones infactibles (tabla 13) y otra sobre la calidad de las soluciónes factibles (tabla 14).

Instancia	PSO	MMC
Caomputación	0	0
Instrumentación I	11	2
Instrimentación II	6	2
Tecnología I	3	3
Tecnología II	2	11

Tabla 13.: Número de soluciones infactibles con MMC

Instancia	PSO	MMC
Caomputación	11	13
Instrumentación I	13.4210	13
Instrimentación II	13.375	13
Tecnología I	13.4074	13
Tecnología II	14.2132	13

Tabla 14.: Calidad de soluciones factibles con MMC

La tabla 15 muestra los resultados de las pruebas que indican que ambas técnicas heurísticas no son estadísticamente diferentes, es decir, la calidad de las soluciones entregadas por ambas técnicas son semejantes.

Prueba	H	P
1	Falso	0.7857
2	Falso	0.0952

Tabla 15.: Resultados

7

CONCLUSIÓN Y TRABAJOS FUTUROS

En este proyecto se implementó una versión discreta de la técnica heurística *PSO* para resolver un problema de empaquetamiento con restricciones de precedencia basado en un información real obtenido de varios planes de estudio de la Universidad Autónoma Metropolitana Unidad Azcapotzalco. Basado en los resultados obtenidos con las instancias seleccionadas, podemos decir que tanto *PSO* como *MMC* representan opciones que generan soluciones competitivas.

Sin embargo, *MMC* demostró superar a PSO en cuanto a tiempo de ejecución por lo que en futuros trabajos se espera que la eficacia de *PSO* se incremente con el uso de topologías, lo que conducirá a una reducción en el número de partículas necesarias y por lo tanto a una reducción en el tiempo de ejecución.

Parte II

APÉNDICE

A

INSTANCIAS

Tronco de Nivelación Académica

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1100033	3				
1111078	4				
1112026	7				
1201008	4				

Tronco General

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1111079	9	1111078			
1111092	3	1111079			
1111081	9	1111079			
1111093	3	1111081	1111092		
1111083	9	1111081			
1112013	9	1112026			
1112027	6	1112026			

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1112028	9	1112027			
1112029	9	1112028			
1112030	9	1112029			
1113046	6	1112028	1111081		
1113084	9				
1113085	3	1113084			
1113086	6	1113084			
1113087	3	1113085			
1151038	7	1112013	1112027		
1151039	7	1151038			
1153001	9	1112029			

Ingeniería en Computación - Tronco Básico Profesional

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1151042	8	1151038			
1121037	12	1151038			
1121040	6	1121037			
1121060	9	1151042	1121037		
1121033	3	1121060			
1151044	8	1151038			
1112017	9	1151038			
1112033	9	1151038			
1152001	9	1151039			
1121038	9	1121060			
1121025	9	1121060			

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1151072	3	1151044			
1151041	8	1151042	1153001		
1112040	9	1112030			
1151040	9	1151042	1112033	1152001	
1121043	12	1121038			
1151018	9	1121025	1121038		
1151047	12	1151072	1151041		
1151048	8	1151072	1151041		
1112034	9	1112033			
1151052	7	1151072			
1124052	9	1111083	1113086		
1151051	9	1112017			
1151050	6	1100040			
1151046	9	1151018	1151047		
1151049	9	1112034			
1100103	3	360			
1100113	18	1100103			

Ingeniería en Computación - Sin Área

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1100040	6	300			
Opt:1100083	6	150			
Opt:1100086	6	150			
Opt:1151054	7	1151048	1151047	1121038	
Opt:1151057	8	1151048	1151047	1121038	

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1136005	6	360			
Opt:1100094	6	150			
Opt:1100084	6	150			
Opt:1100087	6	150			
Opt:1151056	8	1151048			
Opt:1151034	9	1151048			
Opt:1151074	8	1151046	1151047		
Opt:1151058	8	1151054	1151056	1151057	1151074
Opt:1151055	8	1151048	1151047		

Ingeniería Física - Tronco Básico Profesional

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1111013	9	1111081	1112005	1112030	
1111019	9	1137007			
1111043	9	1111090	1111091		
1111044	9	1111043			
1111048	9	1111090	1111091		
1111055	9	1111090			
1111069	3	1111055			
1111085	9	1112029	1112005		

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
1111087	3	1111048			
1111088	3	1111048			
1111094	3	1111083			
1111090	9	1111083	1112030		
1111091	9	1111085	1112015		
1112005	12	1112029	1112013		
1112015	9	1112030			
1112016	6	1112005			
1113069	9	1113046			
1113070	3	1113069			
1122012	9	1112015			
1124001	9	1112030			
1124005	3	1124001			
1132064	3	1133048			
1133048	6	1153001			
1137005	9	1111081	1112030		
1137006	9	1113046			
1137007	9	1112029	1137006		
1154029	9	1153001			
1100037	6	50			
1100038	6	1100037			
1100040	6	280			
1100039	6	1100040			
1100041	6	1100039			
1100106	3	360			
1100116	18	1100106			

Ingeniería Física - Instrumentación I

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1100077	6	150			
Opt:1100078	6	150			
Opt:1100079	6	150			
Opt:1100080	6	150			
Opt:1111054	9	1123040			
Opt:1111058	9	1123016			
Opt:1111060	9	1123016			
Opt:1121037	12	1151038			
Opt:1121040	6	1121037			
Opt:1123016	9	1123040			
Opt:1123021	9	1121037			
Opt:1123045	3	1123040			
Opt:1123040	9	1124001	1124005		

Ingeniería Física - Instrumentación II

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1100077	6	150			
Opt:1100078	6	150			
Opt:1100079	6	150			
Opt:1100080	6	150			
Opt:1111054	9	1123040			
Opt:1111058	9	1123016			

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1111060	9	1123016			
Opt:1121037	12	1151038			
Opt:1121040	6	1121037			
Opt:1123016	9	1123040			
Opt:1123021	9	1121037			
Opt:1123045	3	1123040			
Opt:1123040	9	1124001	1124005		

Ingeniería Física - Tecnología I

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1100077	6	150			
Opt:1100078	6	150			
Opt:1100079	6	150			
Opt:1100080	6	150			
Opt:1111032	9	1111048			
Opt:1111034	9	1111043			
Opt:1133014	9	1145054			
Opt:1141006	3	1146038			
Opt:1145052	6	1145054			
Opt:1145054	9	1112028	1113086	1113087	1113046
Opt:1145056	9	1112030			
Opt:1145057	3	1145056			
Opt:1145071	6	1145054			
Opt:1145072	3	1145071			
Opt:1145091	9	1145052			

Ingeniería Física - Tecnología II

Clave	C	Ser 1	Ser 2	Ser 3	Ser 4
Opt:1100077	6	150			
Opt:1100078	6	150			
Opt:1100079	6	150			
Opt:1100080	6	150			
Opt:1111045	9	1111081	1112005		
Opt:1142025	3	1146038			
Opt:1145001	6	1145054			
Opt:1145050	6	1111032			
Opt:1145052	6	1145054			
Opt:1145054	9	1112028	1113086		
Opt:1145058	9	1137006			
Opt:1145060	9	1145052			
Opt:1145066	9	1145060			
Opt:1146038	9	1145054			

B

RESULTADOS

B.1 RESULTADOS PSO

Instancia: Computación

Corrida	Trimestres	Violaciones1	Tiempo
1	11	0	12.46800017
2	11	0	8.369999886
3	11	0	12.977
4	11	0	16.02899981
5	11	0	11.03600001
6	11	0	7.04399991
7	11	0	13.3599999
8	11	0	16.36100006
9	11	0	15.00800014

Corrida	Trimestres	Violaciones1	Tiempo
10	11	0	6.732000113
11	11	0	10.25200009
12	11	0	79.32299995
13	11	0	8.181999922
14	11	0	13.76999998
15	11	0	12.66599989
16	11	0	5.007999897
17	11	0	13.38400006
18	11	0	10.90500021
19	11	0	12.79700017
20	11	0	3.029999971
21	11	0	13.3269999
22	11	0	29.03999996
23	11	0	69.579
24	11	0	3.667000055
25	11	0	10.74699998
26	11	0	15.02600002
27	11	0	8.15199995
28	11	0	12.90700006
29	11	0	12.55400014
30	11	0	7.082000017

Instancia: Instrumentación I

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	56.31399989
2	16	0	9.833000183
3	14	0	20.4749999
4	13	0	59.94199991
5	14	0	21.60100007
6	14	0	55.51300001
7	13	0	28.45900011
8	13	0	14.2190001
9	13	0	82.97199988
10	13	2	123.927
11	13	0	79.30299997
12	13	0	38.38499999
13	13	2	123.4420002
14	13	2	124.4950001
15	13	1	123.4630001
16	14	0	25.27900004
17	13	0	35.0170002
18	13	2	122.5400002
19	13	0	101.96
20	13	1	122.734
21	13	1	123.5420001
22	13	0	42.11899996
23	14	0	31.6500001

Corrida	Trimestres	Violaciones1	Tiempo
24	13	0	107.0020001
25	13	0	38.70499992
26	16	2	121.8339999
27	13	0	63.28699994
28	13	2	122.2639999
29	13	1	122.0550001
30	17	2	130.0339999

Instancia: Instrumentación II

Corrida	Trimestres	Violaciones1	Tiempo
1	14	0	26.33999991
2	13	2	125.4330001
3	14	0	60.89700007
4	13	0	57.36699986
5	14	0	16.07999992
6	13	0	59.14699984
7	13	0	22.50300002
8	13	2	123.358
9	13	0	82.9000001
10	14	0	95.40200019
11	13	0	108.3410001
12	13	0	62.8829999
13	17	1	121.8729999
14	14	0	117.654
15	13	2	124.3380001
16	13	0	33.11299992
17	14	0	31.94099998
18	13	0	47.16200018
19	13	0	27.46199989
20	13	1	121.3889999
21	14	0	20.64300013
22	13	0	68.81500006
23	14	0	43.92700005

Corrida	Trimestres	Violaciones1	Tiempo
24	13	1	123.392
25	13	0	84.22900009
26	13	0	39.50900006
27	13	0	58.97900009
28	13	0	22.91000009
29	13	0	118.9520001
30	14	0	68.36899996

Instancia: Tecnología I

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	93.52799988
2	14	0	62.2980001
3	14	0	61.18299985
4	14	0	53.31500006
5	13	0	130.2079999
6	13	0	51.57799983
7	14	0	57.02200007
8	13	0	118.523
9	13	0	121.0450001
10	14	0	15.05500007
11	14	0	34.69700003
12	13	0	75.977
13	13	0	35.84500003
14	13	0	43.12600017
15	13	0	104.862
16	14	0	103.8299999
17	13	0	31.80499983
18	14	1	131.8439999
19	14	0	28.32299995
20	13	0	75.48500013
21	13	1	130.974
22	13	0	126.0209999
23	13	0	76.34300017

Corrida	Trimestres	Violaciones1	Tiempo
24	13	1	130.0369999
25	13	0	47.95799994
26	13	0	17.48999977
27	14	0	60.41499996
28	14	0	94.79900002
29	14	0	79.4289999
30	13	0	80.55299997

Instancia: Tecnología II

Corrida	Trimestres	Violaciones1	Tiempo
1	15	0	50.72899985
2	14	0	109.964
3	15	1	128.4879999
4	14	0	117.5309999
5	14	0	108.1849999
6	14	0	35.65499997
7	14	0	47.89100003
8	14	0	88.67700005
9	13	0	70.602
10	14	0	48.15900016
11	17	0	27.35099983
12	14	0	52.76600003
13	15	0	58.85400009
14	14	0	121.586
15	14	0	93.08200002
16	14	0	74.29100013
17	15	0	91.72600007
18	14	0	49.58999991
19	14	0	46.82999992
20	14	0	107.783
21	14	0	66.75699997
22	14	0	84.61199999
23	13	0	127.0249999

Corrida	Trimestres	Violaciones1	Tiempo
24	14	0	117.582
25	15	0	46.34300017
26	14	0	38.64600015
27	14	0	59.22799993
28	14	1	136.7059999
29	15	0	108.52
30	14	0	92.63800001

B.2 RESULTADOS MMC

Instancia: Computación

Corrida	Trimestres	Violaciones1	Tiempo
1	11	0	3.2
2	11	0	4.87
3	11	0	17.783
4	11	0	18.313
5	11	0	4.394
6	11	0	6.092
7	11	0	10.7
8	11	0	8.124
9	11	0	4.331

Corrida	Trimestres	Violaciones1	Tiempo
10	11	0	6.071
11	11	0	1.451
12	11	0	2.371
13	11	0	8.256
14	11	0	3.346
15	11	0	4.486
16	11	0	4.178
17	11	0	4.414
18	11	0	5.001
19	11	0	6.45
20	11	0	1.986
21	11	0	4.182
22	11	0	1.413
23	11	0	4.464
24	11	0	3.282
25	11	0	2.514
26	11	0	8.898
27	11	0	2.526
28	11	0	3.641
29	11	0	2.608
30	11	0	5.908

Instancia: Instrumentación I

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	15.001
2	13	0	3.001
3	13	0	28.33
4	13	0	20.762
5	13	0	20
6	13	1	40.382
7	13	0	14.667
8	13	0	10.359
9	13	0	20.453
10	13	0	7.864
11	13	0	38.478
12	13	0	11.997
13	13	1	41.011
14	13	0	20.569
15	13	0	10.097
16	13	0	10.166
17	13	0	9.567
18	13	0	15.221
19	13	0	7.129
20	13	0	19.894
21	13	0	8.605
22	13	0	17.756
23	13	0	20.677

Corrida	Trimestres	Violaciones1	Tiempo
24	13	0	22.67
25	13	0	17.874
26	13	0	26.594
27	13	0	23.045
28	13	0	13.213
29	13	0	14.048
30	13	0	7.045

Instancia: Instrumentación II

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	26.556
2	13	0	12.677
3	13	0	18.862
4	13	0	15.356
5	13	0	5.207
6	13	0	25.224
7	13	1	40.239
8	13	0	16.831
9	13	0	20.581
10	13	0	38.194
11	13	0	14.222
12	13	0	26.698
13	13	0	10.442
14	13	0	14.47
15	13	0	8.014
16	13	0	15.326
17	13	0	12.874
18	13	0	31.659
19	13	0	13.301
20	13	0	18.504
21	13	0	18.097
22	13	0	27.285
23	13	0	14.982

Corrida	Trimestres	Violaciones1	Tiempo
24	13	0	18.43
25	13	0	17.006
26	13	0	19.51
27	13	1	40.484
28	13	0	17.653
29	13	0	23.107
30	13	0	8.916

Instancia: Tecnología I

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	34.752
2	13	0	30.614
3	13	0	15.038
4	13	1	40.582
5	13	0	12.31
6	13	0	21.824
7	13	0	19.689
8	13	0	20.762
9	13	0	8.691
10	13	0	32.027
11	13	0	22.582
12	13	0	13.023
13	13	0	22.591
14	13	0	23.374
15	13	0	9.554
16	13	0	23.478
17	13	0	38.144
18	13	0	27.397
19	13	1	40.319
20	13	1	41.313
21	13	0	9.434
22	13	0	5.047
23	13	0	29.319

Corrida	Trimestres	Violaciones1	Tiempo
24	13	0	23.539
25	13	0	19.915
26	13	0	18.353
27	13	0	9.797
28	13	0	31.01
29	13	0	19.953
30	13	0	13.681

Instancia: Tecnología II

Corrida	Trimestres	Violaciones1	Tiempo
1	13	0	23.943
2	13	0	8.704
3	13	0	28.923
4	13	0	38.568
5	13	1	40.718
6	13	1	40.721
7	13	0	29.286
8	13	0	22.054
9	13	0	8.73
10	13	0	11.535
11	13	0	20.088
12	13	0	7.064
13	13	1	40.565
14	13	1	40.76
15	13	2	40.878
16	13	0	14.331
17	13	0	25.083
18	13	2	40.762
19	13	1	40.9
20	13	1	40.121
21	13	0	18.372
22	13	0	18.773
23	13	0	27.934

Corrida	Trimestres	Violaciones1	Tiempo
24	13	1	40.261
25	13	0	16.406
26	13	0	36.005
27	13	0	24.102
28	13	1	40.794
29	13	0	31.617
30	13	1	40.591

C

CÓDIGO FUENTE

En las siguientes páginas se muestra un listado con el código fuente desarrollado. Para ejecutarlo, se deberá descargar el intérprete de Python para Windows *PyPy*[1]. Se deberá copiar el código fuente en un archivo llamándolo *ParticleSwarmOptimization.py* y este deberá guardarse en la carpeta de instalación de *PyPy*. Mediante una línea de comandos se deberá navegar hasta la carpeta antes mencionada y escribir lo siguiente:

pypy ParticleSwarmOptimization.py *nombreDeInstancia.txt*

Donde *nombreDeInstancia.txt* es uno de los 4 archivos de texto proporcionados.

1 Descargar de www.pypy.org/download.html

```
#-*- coding: utf-8 -*-
from random import *
from math import *
from copy import deepcopy
import time
import sys

class UEA:
    """Clase que representa una UEA dentro del plan de estudio elegido"""
    def __init__(self, clave, nombre, creditos, ueas_requeridas, creditos_requeridos):
        self.clave = clave
        self.nombre = nombre
        self.creditos = creditos
        self.ueas_requeridas = ueas_requeridas
        self.creditos_requeridos = creditos_requeridos

class Particula:
    """Clase que representa a cada una de las partículas que
    conforman el enjambre"""
    def __init__(self, limites_inferiores, dimension, rango_maximo):
        self.posicion = []
        self.velocidad = []
        self.mejor_posicion = []
        self.calidad_mejor_posicion = 0
        self.calidad = 0
        self.calidad_EC = 0
        self.calidad_RC = 0
        self.calidad_S = 0
        self.calidad_NT = 0
        self.calidad_FN = 0
        self.ueas_violadas = []
        self.inicializar(limites_inferiores, dimension, rango_maximo)

    def inicializar(self, limites_inferiores, dimension, rango_maximo):
        """Inicializa de forma aleatoria la posición y velocidad
        de la partícula"""
        #Inicializar posicion
        self.posicion = [None] * dimension
        for x in xrange(dimension):
            self.posicion[x] = randint(1, rango_maximo)
            if self.posicion[x] < limites_inferiores[x + 1]:
                self.posicion[x] = limites_inferiores[x + 1]
        self.mejor_posicion = self.posicion[:]
        self.calidad_mejor_posicion = 200000
        #Inicializar velocidad
        self.velocidad = [None] * dimension
        for x in xrange(dimension):
            self.velocidad[x] = randint(-3, 3)

def evaluar(posicion, ueas):
    """Calcula la calidad del vector posición que se pasa como parámetro"""
    calidad, calidad_EC, calidad_RC, calidad_S = (0 , 0, 0, 0)
    calidad_NT, calidad_FN = (0, 0)
    ueas_violadas = []
    trimestres = {}
    for i in xrange(1, 19):
        trimestres[i] = 0

    #Cálculo del número de créditos por trimestre
    for x in xrange(1, len(ueas) + 1):
        uea_actual = ueas[x]
```


Figura 1.: Módulos.

```
        trimestres[posicion[x-1]] = trimestres[posicion[x-1]] + uea_actual.creditos

        #Cálculo de la penalización por seriación
        if uea_actual.ueas_requeridas != [0]:
            ueas_relacionadas = [0, []]
            for clave_uea_seriada in uea_actual.ueas_requeridas:
                if posicion[clave_uea_seriada-1] >= posicion[uea_actual.clave-1]:
                    pen = posicion[clave_uea_seriada-1] - posicion[uea_actual.clave-1] + 1
                    calidad = calidad + pen
                    calidad_S = calidad_S + pen
                    if uea_actual.clave not in ueas_violadas:
                        ueas_relacionadas[0] = uea_actual.clave
                    ueas_relacionadas[1].append(clave_uea_seriada)
            if ueas_relacionadas[0] != 0:
                ueas_violadas.append(ueas_relacionadas)

        #Cálculo de la penalización por restricción de créditos
        if uea_actual.creditos_requeridos != 0:
            creditos_acumulados = 0
            for m in xrange(1, posicion[x-1]):
                creditos_acumulados = creditos_acumulados + trimestres[m]

            if creditos_acumulados < uea_actual.creditos_requeridos:
                diferencia = uea_actual.creditos_requeridos - creditos_acumulados
                calidad = calidad + diferencia
                calidad_RC = calidad_RC + diferencia
    #Cálculo de la penalización por el número de trimestres en la solución
    penalizacion = 0
    max_trim = max(posicion)
    penalizacion = abs(max_trim-13)*40
    calidad = calidad + penalizacion
    calidad_NT = calidad_NT + penalizacion

    #Cálculo de la calidad por el exceso de créditos por trimestre
    if trimestres[1] == 0:
        calidad = calidad + 56
        calidad_EC = calidad_EC + 56
    else:
        if trimestres[1] > 56:
            diferencia = trimestres[1] - 56
            calidad = calidad + diferencia*max_trim
            calidad_EC = calidad_EC + diferencia*max_trim

    for h in xrange(2, 19):
        if trimestres[h] > 40:
            diferencia = trimestres[h] - 40
            calidad = calidad + diferencia*max_trim
            calidad_EC = calidad_EC + diferencia*max_trim

    for h in xrange(2, 18):
        if trimestres[h] == 0:
            if trimestres[h + 1] > 0:
                calidad = calidad + 40
                calidad_EC = calidad_EC + 40

    ueas_en_max_trim = 0
    for trimestre in posicion:
        if trimestre == max_trim:
            ueas_en_max_trim += 1
    calidad_FN = 25*max_trim + ueas_en_max_trim
```

```
        calidad += calidad_FN

        return calidad, calidad_EC, calidad_RC, calidad_S, \
        calidad_NT, ueas_violadas, calidad_FN

    def pso(n_corrida, ueas, limites_inferiores, limites_superiores, numero_de_particulas):
        """Función principal que itera a través de todas las partículas, las evalúa y
        extrae la mejor posición conocida por la partícula y la mejor posición conocida
        por todas las partículas, para después realizar la actualización de todo el
        enjambre."""
        numero_de_interaciones = 100001
        f = 1 / 0.000001
        w = 0.7298
        continuar = 1
        print 'Corrida', n_corrida
        contador_global = 0
        suertudas = []
        rango_maximo = 18
        mejor_resultado = None
        while continuar:
            particulas = []
            for y in xrange(numero_de_particulas - len(suertudas)):
                particulas.append(Particula(limites_inferiores, len(ueas), rango_maximo ))

            for j in xrange(len(suertudas)):
                particulas.append(suertudas[j])

            P_global = Particula(limites_inferiores, len(ueas), rango_maximo)

            P_global.calidad = 200000
            clip = lambda n, minn, maxn: max(min(maxn, n), minn)
            for i in xrange(numero_de_interaciones):
                if max(P_global.posicion) <= 12:
                    w = 0.5298
                for q in P_global.ueas_violadas:
                    P_global.posicion[q[0]-1], P_global.posicion[q[1][0]-1] = \
                    P_global.posicion[q[1][0]-1], P_global.posicion[q[0]-1]
                #Ciclo principal
                for particula in particulas:
                    particula.calidad, particula.calidad_EC, particula.calidad_RC, \
                    particula.calidad_S, particula.calidad_NT, particula.ueas_violadas,\
                     particula.calidad_FN = evaluar(particula.posicion, ueas)
                    #Extraer mejor posición conocida por la partícula
                    if particula.calidad <= particula.calidad_mejor_posicion:
                        particula.mejor_posicion = particula.posicion[:]
                        particula.calidad_mejor_posicion = particula.calidad
                    #Extraer la partícula global
                    if particula.calidad <= P_global.calidad:
                        P_global.posicion = particula.posicion[:]
                        P_global.calidad = particula.calidad
                        P_global.calidad_EC = particula.calidad_EC
                        P_global.calidad_RC = particula.calidad_RC
                        P_global.calidad_S = particula.calidad_S
                        P_global.calidad_NT = particula.calidad_NT
                        P_global.ueas_violadas = particula.ueas_violadas[:]
                        P_global.calidad_FN = particula.calidad_FN


                    maximo_trimestre = max(particula.posicion)
                    #Calcular nueva velocidad y posicion
                    for x in xrange(len(ueas)):
```

```
            velocidad = w*particula.velocidad[x] + \
            (randrange(0, 1.49618*f)/f)* \
            (particula.mejor_posicion[x] - particula.posicion[x]) + \
            (randrange(0, 1.49618*f)/f)* \
            (P_global.posicion[x] - particula.posicion[x])
            particula.velocidad[x] = int(round(velocidad))
            particula.velocidad[x] = clip(particula.velocidad[x], \
            - rango_maximo, rango_maximo)
            #Calcular nueva posicion
            particula.posicion[x] = particula.posicion[x] + particula.velocidad[x]

            if particula.posicion[x] < limites_inferiores[x + 1]:
                particula.posicion[x] = limites_inferiores[x + 1]

            if particula.posicion[x] > (maximo_trimestre - \
                limites_superiores[x + 1]):
                particula.posicion[x] = (maximo_trimestre - \
                    limites_superiores[x + 1])
            #Limitar la velocidad
            particula.posicion[x] = clip(particula.posicion[x], 1, rango_maximo)

    #Condiciones de paro para realizar los experimentos
    if P_global.calidad_EC == 0 and \
    P_global.calidad_RC == 0 and \
    P_global.calidad_S == 0:
        if mejor_resultado != None:
            if max(P_global.posicion) < max(mejor_resultado.posicion):
                mejor_resultado = deepcopy(P_global)
                tiempo = time.time()
                iteracion = i+(100*contador_global)
                print iteracion, 'Global -->', mejor_resultado.calidad, \
                mejor_resultado.calidad_EC, mejor_resultado.calidad_RC, \
                mejor_resultado.calidad_S, (mejor_resultado.calidad_NT/40)+13,\
                 '--',mejor_resultado.calidad_FN
        else:
            mejor_resultado = deepcopy(P_global)
            tiempo = time.time()
            iteracion = i+(100*contador_global)
            print iteracion, 'Global -->', mejor_resultado.calidad, \
            mejor_resultado.calidad_EC, mejor_resultado.calidad_RC, \
            mejor_resultado.calidad_S, (mejor_resultado.calidad_NT/40)+13, \
            '--',mejor_resultado.calidad_FN

    if (P_global.calidad_NT/40) == 0 and \
    P_global.calidad_EC == 0 and \
    P_global.calidad_S == 0 and \
    P_global.calidad_RC == 0:
        print i+(100*contador_global), 'Global -->', P_global.calidad, \
        P_global.calidad_EC, P_global.calidad_RC, \
        P_global.calidad_S, (P_global.calidad_NT/40)+13, '--',\
        P_global.calidad_FN
        return time.time(),mejor_resultado, "Exitosa",\
        max(mejor_resultado.posicion),iteracion

    if contador_global == 20:
        if mejor_resultado != None:
            print i+(100*contador_global), 'Global -->', P_global.calidad,\
             P_global.calidad_EC, P_global.calidad_RC, \
            P_global.calidad_S, (P_global.calidad_NT/40)+13, '--',\
            P_global.calidad_FN
            return tiempo,mejor_resultado, "Exitosa",\
```

```
                    max(mejor_resultado.posicion),iteracion
                else:
                    print i+(100*contador_global), 'Global -->', P_global.calidad,\
                     P_global.calidad_EC, P_global.calidad_RC, \
                    P_global.calidad_S, (P_global.calidad_NT/40)+13, '--',\
                    P_global.calidad_FN
                    return time.time(),P_global, "Fallida",max(P_global.posicion),\
                    i+(100*contador_global)

            if i == 100:
                print i+(100*contador_global), 'Global -->', P_global.calidad,\
                 P_global.calidad_EC, P_global.calidad_RC, \
                P_global.calidad_S, (P_global.calidad_NT/40)+13, '--',\
                P_global.calidad_FN
                #Exterminio de partículas
                suertudas.append(deepcopy(P_global))
                rango_maximo = max(P_global.posicion)
                contador_global += 1
                continuar = 1
                break

    print 'saliendo'
    return P_global.posicion, max(P_global.posicion), 0

def cargar_datos(archivo):
    """Crea un diccionario a partir de un archivodonde cada valor
    de este es un objeto de tipo UEA"""
    ueas = {}
    archivo = open(archivo, "r")
    datos = archivo.readlines()
    for registro in datos:
        linea = registro.split('-')
        clave = int(linea[0])
        nombre = linea[1]
        creditos = int(linea[2])
        ueas_requeridas = []
        for x in linea[3].split(';'):
            ueas_requeridas.append(int(x))
        creditos_requeridos = int(linea[4])
        uea = UEA(clave, nombre, creditos, ueas_requeridas, creditos_requeridos)
        ueas[uea.clave] = uea
    return ueas

def revisar_seriacion(uea, ueas_requeridas, ueas):
    """Función que soporta la recursión de la función
    calcular_seriacion"""
    if ueas_requeridas == [0]:
        return
    else:
        for x in ueas_requeridas:
            if x not in uea.ueas_requeridas:
                uea.ueas_requeridas.append(x)
            revisar_seriacion(uea, ueas[x].ueas_requeridas, ueas)

def calcular_seriacion(ueas):
    """Calcula las cadenas completas de seriación"""
    for uea in ueas.values():
        revisar_seriacion(uea, uea.ueas_requeridas, ueas)

def calcular_limites_inferiores(ueas):
    """Calcula el trimestre mínimo en el cual una UEA
```

```
    puede ser ubicada"""
    limites_inferiores = {}
    for uea in ueas.values():
        if uea.ueas_requeridas == [0]:
            limites_inferiores[uea.clave] = 1
        else:
            limites_inferiores[uea.clave] = 2
    for n in xrange(2, 19):
        calcular_limites_i(n, limites_inferiores, ueas)
    return limites_inferiores

def calcular_limites_i(n, limites_inferiores, ueas):
    """Función que soporta la recursión de la función
    calcular_limites_inferiores"""
    for uea_actual in ueas.values():
        if uea_actual.ueas_requeridas != [0]:
            for clave_uea_seriada in uea_actual.ueas_requeridas:
                if uea_actual.creditos_requeridos < \
                ueas[clave_uea_seriada].creditos_requeridos:
                    uea_actual.creditos_requeridos = \
                    ueas[clave_uea_seriada].creditos_requeridos
                if limites_inferiores[clave_uea_seriada] > n-1:
                    limites_inferiores[uea_actual.clave] = n + 1


def calcular_cadenas_de_seriacion(ueas):
    """Enlista todas las posibles cadenas de seriación"""
    cadenas_de_seriacion = []
    for uea_actual in ueas.values():
        if uea_actual.ueas_requeridas != [0]:
            for clave_uea_requerida in uea_actual.ueas_requeridas:
                tmp_list = [uea_actual.clave, clave_uea_requerida]
                calcular_limites_s(tmp_list, ueas, clave_uea_requerida, \
                cadenas_de_seriacion)

    return cadenas_de_seriacion

def calcular_limites_s(tmp_list, ueas, clave_uea_requerida, cadenas_de_seriacion):
    """Función que soporta la recursión de la función
    calcular_cadenas_de_seriacion"""
    if ueas[clave_uea_requerida].ueas_requeridas != [0]:
        for uea in ueas[clave_uea_requerida].ueas_requeridas:
            tmp_list2 = tmp_list[:]
            tmp_list2.append(uea)
            calcular_limites_s(tmp_list2, ueas, uea, cadenas_de_seriacion)
            cadenas_de_seriacion.append(tmp_list)
    else:
        cadenas_de_seriacion.append(tmp_list)

def calc_ls(cadenas_de_seriacion, ueas):
    """Calcula la profundidad de la mayor cadena de seriación
    de cada UEA"""
    limites_superiores = {}
    for uea in ueas.values():
        limites_superiores[uea.clave] = 0

    for cadena in cadenas_de_seriacion:
        for x in xrange(1,len(cadena)):
            if limites_superiores[cadena[x]] < x:
                limites_superiores[cadena[x]] = x
```

```
    return limites_superiores


def main():
    ueas = cargar_datos(sys.argv[1])
    calcular_seriacion(ueas)
    #Se hacen dos pasadas para calcular seriacion debido al orden
    #en el que pueden aparecer definidas las UEA.
    calcular_seriacion(ueas)
    limites_inferiores = calcular_limites_inferiores(ueas)
    limites_superiores = calc_ls(calcular_cadenas_de_seriacion(ueas), ueas)
    numero_de_particulas = 500
    res = {}
    exitosas = 0
    fallidas = 0
    for x in xrange(1,31):
        inicio = time.time()
        tiempo,resultados,estado, trimestres, iteraciones = pso(x, ueas,\
         limites_inferiores, limites_superiores, numero_de_particulas)
        #Analizar el número de violaciones
        violacion_seriacion = 0
        violacion_restriccion_creditos = 0
        violacion_exceso_creditos = 0

        if resultados.calidad_S != 0:
            violacion_seriacion = 1

        if resultados.calidad_RC != 0:
            violacion_restriccion_creditos = 1

        if resultados.calidad_EC != 0:
            violacion_restriccion_creditos = 1

        violaciones = violacion_seriacion + violacion_restriccion_creditos +\
         violacion_exceso_creditos

        res[x] = [trimestres, violaciones, tiempo - inicio]

        print sys.argv[1]
        print 'Trimestres,','Violaciones,','Tiempo'
        for k,v in res.items():
            print v[0],',',v[1],',',v[2]

main()
```

BIBLIOGRAFÍA