Universidad de Granada

E.T.S Ingeniería Informática y de Telecomunicación
Grado en Ingeniería Informática
Departamento de Ciencias de la Computación e Inteligencia Artificial

Metaheurísticas - MH
Grupo A1 (Miércoles 17:30 - 19:30)
Curso 2019/2020

Práctica Final:
Estudio sobre Metaheurística Original inspirada en la adaptación de estrategias en juegos competitivos (Competitive Adaptation)

Javier Rodríguez Rodríguez
78306251Z
e.doblerodriguez@go.ugr.es

1. Índice

2. Inspiración, motivación y justificación

Actualmente un subgénero importante de los videojuegos es el de los denominados juegos competitivos, un tipo de juego multijugador donde la intención es permitir a los jugadores competir entre ellos. No obstante, este concepto es sumamente antiguo y engloba múltiples actividades, pues juegos de cartas coleccionables, juegos de mesa o incluso deportes pueden ser calificados como juegos competitivos. Lo único que los agrupa es el sentido de competitividad y la búsqueda de la victoria ante adversarios como foco principal.

Un concepto común a todos estos juegos es el metajuego, los conceptos que definen la mejor forma o estrategia de jugar un determinado juego y maximizar las posibilidades de ganar. El metajuego normalmente se va desarrollando de forma orgánica a medida que los jugadores se familiarizan más y más con el juego específico, pero en los casos más complejos llegan a surgir figuras dedicadas a analizar el juego desde una perspectiva más teórica o abstracta para detectar posibles mejoras competitivas sobre el metajuego establecido. Sin duda, encontrar mejoras en el metajuego es fundamental y de muchísimo interés para todos aquellos que participan en juegos competitivos, pues su éxito depende de qué tanto lo entiendan y lo consideren en sus acciones. Ser capaz de obtener y mantener una estrategia exitosa da lugar, en muchos casos, a carreras capaces de generar elevadas cantidades de dinero y reconocimiento mundial, sin duda no se trata de un interés trivial.

¿Qué interés tiene esto dentro del campo de las metaheurísticas? Para entenderlo, es necesario analizar las similitudes entre los conceptos. En primer lugar, la presencia del prefijo Meta, prefijo que denota una abstracción de orden mayor sobre el concepto que le sigue, presenta en sí misma una similitud fonética y conceptual entre metajuego y metaheurística: son procesos que extienden y completan su concepto base. En el caso de las metaheurísticas, se sabe que esta extensión se trata del desarrollo de técnicas de propósito general que, con cierto grado de adaptación, son capaces de proporcionar respuestas adecuadas para múltiples problemas de optimización. Sin embargo, en el caso del metajuego, quedó aclarado previamente que dependen de las reglas y el funcionamiento de cada juego, su desarrollo es específico y no se trata de reglas de propósito general. Entonces, ¿por qué es interesante?

La respuesta se halla en el mismo término que engloba a todos estos juegos: la competitividad. Independientemente del tipo de juego y las reglas que cuente, el interés último se encuentra en ganar y, por esta razón, sus jugadores, de forma consciente o inconsciente, obedecen a la tendencia de imitar y adaptar estrategias preexistentes que den buenos resultados. Es ésta la componente principal con la cual es posible trazar múltiples paralelismos con el concepto de metaheurísticas: una técnica de optimización generalizada capaz de adaptarse de forma relativamente sencilla a una variedad de problemas y obtener resultados adecuados.

Se podría pensar que un proceso social que, en su esencia más resumida, consiste en seguir modas, podría ser una idea poco competitiva a la hora de emplearla dentro del área de las metaheurísticas, pero la historia dice lo contrario: tanto los videojuegos más modernos como los deportes más antiguos presentan este suceso en múltiples momentos de su historia, e incluso algunos modifican sus reglas en función de este comportamiento. El surgimiento de estrategias como el Catenaccio italiano, el Fútbol Total holandés y el Tiquitaca español le han permitido a sus selecciones de fútbol posicionarse en las primeras posiciones de mundiales de fútbol a lo largo de la historia, y es el proceso de imitación y desarrollo por parte del resto de los equipos lo que lleva a la evolución del juego. Es este suceso, repetido en tantos tipos de juegos tan variados a lo largo de la historia, lo que impulsa al desarrollo de una metaheurística que se inspire en éste.

Sin embargo, antes de dar inicio a la implementación, hay que dar respuesta a otra interrogante fundamental. Ya quedó establecido de qué se trata la inspiración, de dónde surge la motivación y por qué se espera que dé buenos resultados pero, ¿qué la diferencia de las metaheurísticas previamente estudiadas? La respuesta radica en la consideración de un elemento que hasta ahora había se había ignorado: el orden de calidad entre las estrategias existentes en un momento dado. Las estrategias poblacionales estudiadas enfrentan a soluciones, pero nunca llega a establecer un orden total en la población. La idea de esta metaheurística es aprovechar ese elemento no utilizado para plantear un podio, un ránking de soluciones que sean la base para la adaptación de nuevas, y así avanzar hasta alcanzar las soluciones deseadas. En este podio es fundamental el orden, así, retomando el ejemplo de estrategias de fútbol, el Fútbol Total

holandés, si bien sumamente interesante, no llegó a alzarse campeón del Mundial de Fútbol, a pesar de ser finalista dos veces consecutivas. En este sentido, a pesar de que claramente se percibe como una estrategia de calidad, se sabe que posiblemente no sea la mejor, pues de serlo vencería a todos sus adversarios, y por esta razón quizás no tuvo el desarrollo tan abrumador que se podría esperar.

Por supuesto, utilizar análisis basado en resultados es más complejo en la vida real, pues la incorporación de factores estocásticos y humanos, así como la imposibilidad de establecer un orden estricto de calidad hace que resulte altamente complejo este tipo de análisis, pero por suerte, dentro del universo de las metaheurísticas, tales elementos se pueden abstraer con facilidad en elementos más sencillos, dando lugar a implementaciones como la que se presenta en este trabajo, capaces de incorporar la estrategia diferenciadora que la justifica: la exploración basada en posiciones del podio.

3. Descripción del problema

El problema propuesto consiste en el agrupamiento (clustering) de instancias de datos no etiquetadas que están sujetos a un conjunto de restricciones de instancia de tipo Must-Link (ML) o Cannot-Link (CL). Estas restricciones se consideran débiles, es decir, no son limitantes a una solución, pero el cumplimiento de estas es un factor importante de calidad de la solución.

Este problema, perteneciente al campo de aprendizaje semi-supervisado, es NP-Completo, y por tanto, resulta imposible obtener un algoritmo que halle consistentemente la solución óptima en un tiempo razonable. Entendiendo esta limitación, y razonando dentro del campo de la metaheurística, se compararán los algoritmos basados en trayectorias pertenecientes a la Práctica 3 de la asignatura con la metaheurística original basada en adaptación de estrategias competitivas presentada, así como sus mejoras correspondientes.

La comparación entre los algoritmos estudiados se establecerá en dos niveles distintos: la calidad de la solución y la eficiencia temporal. La calidad debe medir adecuadamente qué tan parecidos son realmente los datos que se agrupan en un mismo cluster y cuántas restricciones incumple, mientras que la eficiencia temporal debe medir el tiempo requerido para dar una solución que el criterio del algoritmo considere suficiente.

4. Aplicación de los algoritmos

Para aplicar los algoritmos descritos, primero que nada, es necesario describir las estructuras de datos que contienen toda la información. Los datos de entrada, las instancias y las restricciones, son provistas en forma de archivos en texto plano cuyos nombres identifican el conjunto de datos de donde se extrajeron, y en el caso de ser restricciones, el porcentaje de relaciones restringidas. Estos ficheros se leen y almacenan en una estructura de matriz: para las instancias, cada fila corresponde a una instancia, mientras que las columnas corresponden a las distintas dimensiones de cada instancia; para las restricciones, es una matriz cuadrada simétrica donde cada valor [i,j] $\in \{-1,0,1\}$ corresponde a la restricción entre la instancia i y la instancia j. Cada valor representa un tipo de restricción, siendo -1 una restricción CL, 1 una restricción ML y 0 ninguna restricción.

La solución del problema, para un conjunto de instancias y restricciones dado, vendrá dado en un vector de tantos elementos como instancias de datos haya, y que en cada posición contendrá el índice del cluster al que pertenece la instancia en dicha fila en la matriz de datos: es decir, Solución[i] contendrá el número de cluster al que pertenece Datos[i].

Para la medición de la calidad se emplea una función objetivo, cuyo valor ha de expresar la tasa de diferencia entre los datos de un mismo cluster y la cantidad de restricciones incumplidas, y por tanto, se ha de minimizar mientras de mayor calidad sea la solución. Utilizaremos la siguiente función

$$\textbf{Objetivo} = C_{General} + \textbf{Infeasibility} * \lambda$$

donde:

- $C_{General}$ corresponde a la desviación general de la solución, es decir, el valor medio de desviación de cada cluster. La desviación de un cluster corresponde al valor medio de las distancias entre su centroide asociado y cada una de las instancias contenidas en el cluster. Es decir:

$$(1)\ C_{General} = \sum_{i=1}^{k} C_i / k$$

$$(2)\ C_i = \sum_{j=1}^{ni} |\mu_i - x_j| / j$$

donde k sea la cantidad de clústeres, μ_i el centroide del cluster i, x_j el dato j en el cluster y ni el número de datos en el cluster i.

- Infeasibility es un número entero que corresponde a la cantidad de restricciones incumplidas

- λ es un factor de escala, con el propósito de darle peso suficiente a la infeasibility y que se valore correctamente respecto al valor de $C_{General}$. Un valor general que se considera correcto es un valor mayor a la distancia máxima entre un par de instancias, seleccionando nosotros el techo de dicha distancia, entre la cantidad de restricciones. Es decir:

  $$(3)\ \lambda = \text{techo}(D_{max}) / |R|$$

  siendo |R| el número de restricciones y D_{max} la distancia máxima entre dos instancias. Importante destacar, de cara a la implementación, que en la contabilización de restricciones debemos evitar contabilizar cada restricción dos veces (debido a la simetría de la matriz)

Para la medición de la eficiencia temporal, consideraremos como métrica el tiempo de ejecución de ambos algoritmos en condiciones similares (mismo equipo bajo carga similar).

Por último, es necesario establecer, dado que ambos algoritmos contienen instrucciones estocásticas, es necesario especificarle a los algoritmos un generador de números aleatorios que, para una semilla dada, sea capaz de replicar los valores generados.

Entre estas instrucciones estocásticas, una presente en todas las técnicas analizadas es la generación de una solución inicial, totalmente aleatoria. Sin embargo, esta solución ha de cumplir con la regla de validez de toda solución: que ningún clúster quede vacío.

Para ello, la estrategia es la siguiente

Solución Inicial(k, tamaño):
# k es la cantidad de clústeres
# i_j es un valor aleatorio ∈ {0, 1, …, k-1}
Solución = {0, 1, …, k-1} ∪ {i_k, i_{k+1}, …, $i_{tamaño-1}$}
Solución = Barajar(Solución)
Retornar Solución

Este pequeño trozo de código asegura que todos los clústeres tengan por lo menos un elemento asignando, de partida, los primeros k elementos a

cada clúster respectivamente, y por tanto asignando un elemento a cada clúster. El resto de tamaño - k elementos se asignan aleatoriamente entre los valores de cada clúster, sin ningún tipo de cuidado adicional, pues ya aseguramos la condición de validez. Finalmente, para evitar que las primeras k asignaciones de la solución sean constantes en cada llamada, aplicamos una operación de barajado sobre el vector, asegurando así verdadera aleatoriedad sin perder la certeza de validez.

Para implementar el cálculo antes descrito de la función objetivo, se utiliza el siguiente algoritmo:

```
Evaluación(solución, datos, ml, cl, k, λ):
centroides[k, longitud(datos)] = 0
Para i = 0...k-1:
     centroides[i] = Media(datos en cluster i)
desv_general = 0
Para i = 0...k-1:
     desv_general += Media(Distancia(Centroide[i], dato en cluster
i) / k
infeasibility = |Parejas ML en clústeres distintos| + |Parejas CL en
mismo cluster|
objetivo = desv_general + (infeasibility * λ)
Retornar desv_general, infeasibility, objetivo
```

5. Implementación. Métodos de búsqueda

En esencia, el algoritmo se inspira en los algoritmos poblacionales para su definición inicial: una población de soluciones aleatorias. Así como en los algoritmos genéticos se utilizaban términos específicos como gen y cromosoma para referir a estas soluciones y sus componentes, la población se denominará torneo, y cada solución, una estrategia. Los elementos de cada estrategia simplemente se denominarán valores o componentes de dicha estrategia.

Respecto a los genéticos, la diferencia fundamental radica en ignorar completamente el operador de cruce. A nivel inspiracional, combinar pedazos de múltiples estrategias tiende a tener poco éxito, pues las estrategias rara vez se pueden considerar como una suma de sus partes más que como un único concepto total. En la realidad, cada jugador que adopta y adapta una estrategia tiende a tratar de determinar cuáles elementos son los que menos aportan a la estrategia total y cambiarlos por unos nuevos en búsqueda de obtener un mejor resultado conservando el núcleo que hizo que la estrategia fuera poderosa e interesante en primer lugar. Esto se implementa a través de un operador de mutación fuerte, que reemplaza una selección de valores de la estrategia por una nueva. Esta selección no ha de ser consecutiva y su tamaño es variable. La única restricción a la que está sometida es la de mantener la validez de la estrategia (en el problema en cuestión, no dejar clústeres vacíos) y que no se varíe más de la mitad de la solución, pues en su caso se considera que deja de ser una adaptación de la estrategia original.

La implementación del operador de mutación descrito es la siguiente

```
Mutación(estrategia, máx_variación, k):
tamaño_var = Random(1, máx_variación)
elemAVariar[tamaño_var] = Random(tamaño(estrategia))
resto = (0..tamaño(estrategia)-1) excepto aquellos en elemAVariar
clústeres_vacíos = clústeres que no aparecen en estrategia[resto]
segmento_mutado = clústeres_vacíos ∪ Random(0, k-1) hasta que tamaño(segmento_mutado) = tamaño_var
Barajar(segmento_mutado)
```

```
estrategia[elemAVariar] = segmento_mutado
```

Como vemos, la idea es seleccionar aleatoriamente una determinada cantidad de índices y reasignarlas a nuevos clústeres, controlando solamente que la solución resultante sea válida (no hayan clústeres vacíos).

La razón por la que este operador de mutación se considera más fuerte que los estudiados previamente radica en la cantidad de elementos que muta y la doble componente aleatoria: mientras en los operadores de mutación por segmento fijo los elementos alterados siempre estaban consecutivos, y en algunos casos el tamaño del segmento no variaba, en este caso tanto la cantidad de elementos a mutar como el orden de éstos es estocástico. Sin embargo, visto de forma individual no parece sumamente beneficioso, precisamente porque no controla de ninguna forma la calidad de la solución. Tal control recae en manos del mecanismo de explotación por podio de la metaheurística.

Este mecanismo consiste en evaluar todo el torneo y determinar las mejores 3 soluciones, las cuales se repetirán hasta tomar un determinado porcentaje del tamaño total del torneo, este porcentaje dependiendo de su posición, de forma tal que el primer lugar ocupa el 50% del torneo, el segundo el 30% y el tercero solo el 10%. A cada una de estas copias se les aplica el operador de mutación, de forma tal que se represente cada una de las distintas alteraciones realizadas a las estrategias que previamente resultaron ganadoras. Para asegurar la permanencia y obtención final de la mejor solución encontrada, aquella estrategia que obtenga el primer lugar en un torneo dado se conservará y participará en el siguiente sin ningún tipo de modificación, es decir, existe elitismo dentro de cada torneo. Este comportamiento es coherente con la inspiración, pues normalmente un jugador que tiene éxito con una estrategia no la modifica hasta que pierde su ventaja competitiva. Como dice el dicho, “si no está roto, no lo repares”. El 10% restante corresponde a la introducción de nuevas soluciones aleatorias, que procura ofrecer una componente de exploración más amplia y representar el surgimiento de estrategias sin precedente alguno que, si bien es una rara ocurrencia, llega a pasar.

Finalmente, el algoritmo culminará tras evaluar una determinada cantidad de veces la función objetivo, es decir, realizar

cierta cantidad de torneos, tras lo cual simplemente se indican los datos de la solución ganadora de este último. Como existe elitismo, se cuenta con la certeza que tal solución es la mejor encontrada durante toda la ejecución.
En pseudocódigo, la metaheurística resultante es la siguiente:

```
Adaptación Competitiva(k, tamaño_s, tamaño_torneo, máx_evals):
tamaño_podio = 3
adoptadores = (tamaño_torneo*0.5-1, tamaño_torneo*0.3, tamaño_torneo*0.1)
tamaño_nuevos = tamaño_torneo*0.1
evals = 0
torneo[tamaño_torneo] = Solución Inicial(tamaño_s, k)
puntajes[tamaño_torneo] = Evaluación(i) para todo i en torneo
evals += tamaño_torneo
podio = Índices de los 3 menores valores de objetivo en puntajes, ordenados.
puntajes = puntajes[podio]
podio = torneo[podio]
Mientras evals < máx_evals:
      torneo = podio[0] //El ganador del torneo anterior
      nuevos[tamaño_nuevos] = Solución Inicial(tamaño_s, k)
      torneo = torneo ∪ nuevos
      Para i = 0..tamaño_podio -1:
            torneo = torneo ∪ Mutación(podio[i], tamaño_s/2)
      puntajes = puntajes[0] ∪ Evaluación(i) para todo i en torneo
      evals += tamaño_torneo - 1
      podio = Índices de los 3 menores valores de objetivo en puntajes, ordenados.
      puntajes = puntajes[podio]
      podio = torneo[podio]
Retornar podio[0], puntajes[0] // El ganador final
```

6. Hibridación memética. Búsqueda Local

En el apartado anterior detallamos el diseño e implementación de un mecanismo de mutación que replique la multitud de variaciones y adaptaciones que, en la realidad, los jugadores aplican sobre las estrategias que deciden emplear. Si bien la aleatoriedad intrínseca del operador de mutación permite un elevado poder de exploración, razón principal por la que se decidió utilizar, la realidad es que los jugadores no plantean cambios aleatorios. En su lugar, aplican su conocimiento sobre el juego para determinar los que, desde su perspectiva, son las mejores optimizaciones a una estrategia que ya es prometedora. Como se menciona en la justificación de esta metaheurística, introducir conceptos específicos de cada juego resulta imposible y va en contra del lineamiento general planteado. Sin embargo, a nivel de implementación es posible aprovechar que la función objetivo es un valor numérico constante y abstraer este conocimiento en forma de un método de optimización a través de la explotación del entorno: la Búsqueda Local.

Esta mejora es el equivalente memético a la inspiración genética original, y plantea sustituir el operador de mutación por un operador de Búsqueda Local que imite el comportamiento de la mutación, pero con la idea de conocimiento, al cual, para diferenciarlo de la mutación original y del algoritmo de Búsqueda Local utilizado como metaheurística en sí misma en la Práctica 1, se denominará Refinamiento.

Primero, la implementación de este operador es la siguiente:

```
Refinamiento(estrategia, puntaje, máx_variación, k, máx_evals, datos, ml, cl, λ):
tamaño_var = Random(1, máx_variación)
evals = 0
elemAVariar[tamaño_var] = Random(tamaño(estrategia))
Mientras (no mejor y evals < máx_evals):
     cluster_unico = Clústeres con un solo elemento
     elementos_posibles = Elementos en elemAVariar cuyo cluster no esté en cluster_unico
     vecindario = Pares en elementos_posibles x clústeres tales que el clúster del vecino sea distinto al actual
     Para cada i, j ∈ vecindario:
          vecino = Copia(estrategia)
          vecino[i] = j
          vecino_gd, vecino_i, vecino_o = Evaluación(vecino,
```

```
datos, ml, cl, λ)
          Si vecino_o < puntaje[2]:
               estrategia[i] = j
               puntaje = (vecino_gd, vecino_i, vecino_o)
               mejor = False
               Salir del bucle
Retornar estrategia, puntaje, evals
```

Al detallarla queda más claro que, si bien se trata en su mayor parte de una búsqueda local clásica, más robusta que la búsqueda local suave estudiada en la Práctica 2, se conserva la elección aleatoria de hasta tamaño/2 índices, que serán los únicos sobre los que se considerará la búsqueda local, pues son los únicos datos considerados en el vecindario.

La razón de esta decisión se fundamenta en los mismos elementos que en la utilización de este esquema en la mutación simple: en la adaptación cada jugador contempla un cierto trozo de la estrategia como el menos importante, donde hay más espacio para mejora, que es en el que centra sus esfuerzos. Además, como no super la mitad de la estrategia, se considera que aún conserva suficiente información de la estrategia original para ser una variación de ésta y no una estrategia completamente nueva.

Una de las razones por las que se prefirió utilizar la búsqueda local trabajada en vez de la suave es que, con el cambio antes descrito y una cantidad de iteraciones suficientemente baja (500), los tiempos de convergencia son manejables manteniendo calidad en la solución. Esto se analizará con más detalle en el siguiente apartado. Por otro lado, la búsqueda local planteada tiene como beneficio la certeza que los resultados devueltos son, cuanto menos, similares a los originales, razón por la cual es posible abandonar el esquema de elitismo de la implementación original, ya que por esta propiedad la mejor solución nunca va a desaparecer.

Por último, para contrarrestar el incremento en poder de cómputo y tiempo requerido para realizar estas búsquedas locales, y aprovechando que con el nuevo operador de refinamiento soluciones en un mismo entorno tienden a converger hacia un mismo óptimo local, se contempla un mecanismo de convergencia rápida, que permita dar una solución antes de que se realicen todas las evaluaciones. Esta condición, denominada “metajuego resuelto” por la expresión utilizada cuando en los juegos competitivos se considera

que ya no hay más optimización posible y que se han contemplado todas las posibles variables, se expresa cuando las tres estrategias que forman parte del podio de un torneo dado son iguales. En este caso, el siguiente torneo tendría exclusivamente alteraciones de tal estrategia, y el hecho de que en el torneo anterior convergen a la misma secuencia da a entender que no hay más optimización por realizar. Esta condición de salida permite acelerar notablemente la obtención de resultados conservando el poder de la hibridación memética.

Por último, la implementación resultante de este esquema es la siguiente:

```
Adaptación Competitiva Refinada(k, tamaño_s, tamaño_torneo, máx_evals):
tamaño_podio = 3
adoptadores = (tamaño_torneo*0.5, tamaño_torneo*0.3, tamaño_torneo*0.1)
tamaño_nuevos = tamaño_torneo*0.1
evals = 0
torneo[tamaño_torneo] = Solución Inicial(tamaño_s, k)
puntajes[tamaño_torneo] = Evaluación(i) para todo i en torneo
evals += tamaño_torneo
podio = Índices de los 3 menores valores de objetivo en puntajes, ordenados.
puntajes = puntajes[podio]
podio = torneo[podio]
meta_resuelto = False
Mientras evals < máx_evals y no meta_resuelto:
      torneo = podio[0] //El ganador del torneo anterior
      nuevos[tamaño_nuevos] = Solución Inicial(tamaño_s, k)
      torneo = torneo ∪ nuevos
      Para i = 0..tamaño_podio -1:
            torneo, puntajes, t_evals = torneo, puntajes ∪ Refinamiento(podio[i], puntajes[i], tamaño_s/2, k, máx_evals, datos, ml, cl, λ)
            evals += t_evals
      podio = Índices de los 3 menores valores de objetivo en puntajes, ordenados.
      puntajes = puntajes[podio]
      podio = torneo[podio]
      Si podio[i] == podio[j] ∀ i,j ∈ [0, tamaño_podio-1]:
            meta_resuelto = True
Retornar podio[0], puntajes[0] // El ganador final
```

Como vemos, el esquema es casi idéntico al original, tan solo

cambia la forma en la que se contabilizan las evaluaciones, para contemplar las realizadas dentro de la búsqueda local, el no requerimiento de conservar la solución ganadora como parte de elitismo (pues sus alteraciones serán, cuanto menos, igual de buenas) y la consideración de la condición de meta resuelto como condición alternativa de salida si se considera que ya se convergió en una solución.

7. Procedimiento para desarrollar la práctica

Todo el proceso de procesamiento de datos, implementación de algoritmos y obtención, cálculo y almacenamiento de los resultados finales fue hecho en el lenguaje de programación Python 3.7.6, a través de su distribución especializada para ciencia de datos, Anaconda. La razón de la elección de este lenguaje es por ser uno de los más utilizados y destacados en problemas de aprendizaje, por lo que se consideró una buena oportunidad para practicar su uso y sintaxis, y en particular, el uso relacionado al paquete de cómputo científico NumPy, una de las principales ventajas del lenguaje contra sus competidores y la principal herramienta detrás de todas las implementaciones.

Las ejecuciones fueron realizadas sobre una distribución Ubuntu 18.04 contenida en Windows Subsystem for Linux (WSL), en un equipo con sistema operativo Windows 10 1909.

El código proporcionado se decidió dejar de lado, pues no se consideró necesario implementar el código en Python cuando NumPy provee las herramientas de randomización necesarias.

El proceso de desarrollo de la práctica se hizo de forma lineal, aprovechando los mecanismos ya implementados en las prácticas anteriores: los fragmentos de código para la lectura y ejecución de los datos, así como de evaluación de la función objetivo los tomamos directamente de las prácticas anteriores, luego implementamos la búsqueda local optimizando ligeramente el código de la práctica 1, y finalmente implementamos los nuevos algoritmos en orden de aparición en la documentación, depurando cada uno según se terminaba.

- **Manual de ejecución:**

Es requisito necesario para la ejecución de la práctica que el equipo donde se ejecuten contenga Python 3.4 en adelante, así como el paquete NumPy 1.17. Además, el fichero PFinal.py debe estar en el mismo directorio que un directorio llamado “Instancias y Tablas PAR 2019-20”, que contenga los ficheros de datos y restricciones. Estos ficheros deben ser los contenidos en el fichero comprimido “nuevos conjuntos de datos PAR 2019-20.zip” disponible en la plataforma PRADO.

Una vez se tengan todos los requisitos, tan solo es necesario indicar por terminal la ejecución del script, mediante la sintaxis

> <path a Python> PFinal.py

Durante la ejecución no imprimirá información por pantalla, pues todos los resultados estarán contenidos en el fichero “solutions_PFinal.txt” que se creará en el mismo directorio que contiene el script.

8. Análisis de resultados

Para los resultados, se compararán los resultados de la Práctica 3, es decir, los algoritmos basados en trayectorias y los algoritmos Greedy y Búsqueda Local originales contra la metaheurística original detallada en este documento, así como su hibridación memética. No se entra en detalle sobre los algoritmos previos, pues su análisis e implementación al detalle se encuentra en el documento asociado a su respectiva práctica.

Por fallos de implementación de la Práctica 2, los resultados de ésta no serán considerados.

Los resultados anteriores y nuevos son obtenidos a partir de una ejecución del programa siguiendo las instrucciones anteriores, generando la siguientes tablas de resultados. En todos los casos las semillas correspondientes a cada ejecución son 1, 112, 241, 27, 472, y el tiempo viene expresado en segundos.

Resultados obtenidos por el algoritmo de Adaptación Competitiva Básico (BCA) en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	0.00	0.67	28.44	38.61	1204.00	70.92	51.21
Ejecución 2	0.67	5.00	0.71	28.02	40.42	1108.00	70.15	49.85
Ejecución 3	0.67	0.00	0.67	27.70	38.68	1137.00	69.19	49.65
Ejecución 4	0.68	7.00	0.73	28.29	39.70	1089.00	68.92	50.17
Ejecuión 5	0.67	7.00	0.72	29.28	39.81	1130.00	70.13	50.65
Media	0.67	3.80	0.70	28.34	39.45	1133.60	69.86	50.31

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	27.26	14.69	62.00	16.98	29.46
Ejecución 2	0.77	17.00	0.91	26.74	14.26	112.00	18.40	28.85
Ejecución 3	0.72	0.00	0.72	27.22	14.27	114.00	18.48	28.99
Ejecución 4	0.72	0.00	0.72	27.47	14.08	99.00	17.74	29.18
Ejecuión 5	0.73	5.00	0.77	27.84	14.52	117.00	18.84	30.28
Media	0.73	4.40	0.77	27.31	14.36	100.80	18.09	29.35

Resultados obtenidos por el algoritmo de Adaptación Competitiva Básico (BCA) en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.68	12.00	0.72	28.61	39.11	2355.00	70.71	51.32
Ejecución 2	0.69	52.00	0.88	27.46	38.49	2586.00	73.18	52.10
Ejecución 3	0.67	0.00	0.67	27.60	36.90	2511.00	70.59	52.34
Ejecución 4	0.68	33.00	0.80	28.03	38.12	2226.00	67.99	51.55
Ejecuión 5	0.67	0.00	0.67	28.21	39.33	2308.00	70.30	52.45
Media	0.68	19.40	0.75	27.98	38.39	2397.20	70.55	51.95

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.74	11.00	0.78	28.06	15.84	137.00	18.37	30.27
Ejecución 2	0.74	15.00	0.80	28.11	13.98	282.00	19.19	30.27
Ejecución 3	0.76	30.00	0.88	27.60	14.53	88.00	16.15	32.15
Ejecución 4	0.74	12.00	0.78	27.61	12.85	279.00	18.01	31.27
Ejecuión 5	0.74	16.00	0.80	27.69	13.69	446.00	21.93	32.62
Media	0.74	16.80	0.81	27.82	14.18	246.40	18.73	31.32

Resultados obtenidos por el algoritmo de Adaptación Competitiva Refinado (RCA) en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	***Tasa_inf***	***Agr.***	**T**	***Tasa_C***	***Tasa_inf***	***Agr.***	**T**
Ejecución 1	0.67	0.00	0.67	8.31	22.21	71.00	24.11	177.36
Ejecución 2	0.67	0.00	0.67	8.10	21.08	118.00	24.25	219.22
Ejecución 3	0.67	0.00	0.67	8.48	21.33	79.00	23.45	211.21
Ejecución 4	0.67	0.00	0.67	8.74	22.06	69.00	23.91	188.86
Ejecuión 5	0.67	0.00	0.67	8.77	21.93	58.00	23.48	223.60
Media	0.67	0.00	0.67	8.48	21.72	79.00	23.84	204.05

	Rand				Newthyroid			
	Tasa_C	***Tasa_inf***	***Agr.***	**T**	***Tasa_C***	***Tasa_inf***	***Agr.***	**T**
Ejecución 1	0.72	0.00	0.72	7.66	10.90	97.00	14.48	17.10
Ejecución 2	0.73	7.00	0.79	7.77	13.83	6.00	14.06	17.33
Ejecución 3	0.72	0.00	0.72	8.70	13.83	6.00	14.06	18.60
Ejecución 4	0.72	0.00	0.72	7.74	13.83	6.00	14.06	15.26
Ejecuión 5	0.72	0.00	0.72	8.20	13.83	6.00	14.06	14.79
Media	0.72	1.40	0.73	8.02	13.25	24.20	14.14	16.61

Resultados obtenidos por el algoritmo de Adaptación Competitiva Refinado (RCA) en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	0.00	0.67	9.52	22.07	131.00	23.82	163.51
Ejecución 2	0.67	0.00	0.67	7.67	21.89	132.00	23.66	195.67
Ejecución 3	0.67	0.00	0.67	8.16	21.84	160.00	23.99	177.71
Ejecución 4	0.67	0.00	0.67	7.58	22.00	141.00	23.90	192.32
Ejecuión 5	0.67	0.00	0.67	7.95	21.68	160.00	23.83	174.12
Media	0.67	0.00	0.67	8.18	21.90	144.80	23.84	180.66

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	9.78	10.87	235.00	15.21	16.07
Ejecución 2	0.72	0.00	0.72	9.44	14.29	0.00	14.29	14.75
Ejecución 3	0.72	0.00	0.72	9.07	14.29	0.00	14.29	14.22
Ejecución 4	0.72	0.00	0.72	7.61	14.29	0.00	14.29	14.87
Ejecuión 5	0.72	0.00	0.72	8.02	14.29	0.00	14.29	13.67
Media	0.72	0.00	0.72	8.78	13.60	47.00	14.47	14.72

Resultados globales en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
BL	0.67	0.00	0.67	0.46	22.10	91.00	24.54	16.87
ES	0.67	0.00	0.67	2.48	21.93	87.60	24.28	13.36
BMB	0.67	0.00	0.67	3.77	21.63	162.40	25.98	56.28
ILS	0.67	0.00	0.67	1.83	20.93	94.60	23.47	44.87
ILS-ES	0.67	0.00	0.67	17.76	45.36	1727.00	91.69	12.08
ILS-ES Adapt.	0.67	0.00	0.67	5.39	21.64	115.20	24.73	37.44
COPKM	0.67	4.40	0.70	0.02	37.63	291.20	45.44	1.13
BCA	0.67	3.80	0.70	28.34	39.45	1133.60	69.86	50.31
RCA	0.67	0.00	0.67	8.48	21.72	79.00	23.84	204.05

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
BL	0.72	0.00	0.72	0.39	11.48	79.60	14.42	1.09
ES	0.72	0.00	0.72	1.69	12.66	42.20	14.21	5.61
BMB	0.72	0.00	0.72	3.29	13.83	6.00	14.06	7.33
ILS	0.72	0.00	0.72	1.77	12.07	59.40	14.27	4.52
ILS-ES	0.72	0.00	0.72	16.00	13.12	1085.20	53.22	3.10
ILS-ES Adapt.	0.72	0.00	0.72	3.31	11.48	79.20	14.41	5.30
COPKM	0.71	1.80	0.73	0.02	14.98	78.40	17.88	0.04
BCA	0.73	4.40	0.77	27.31	14.36	100.80	18.09	29.35
RCA	0.72	1.40	0.73	8.02	13.25	24.20	14.14	16.61

Resultados globales en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
BL	0.67	0.00	0.67	0.37	21.92	159.00	24.05	15.22
ES	0.67	0.00	0.67	4.56	21.89	166.80	24.13	43.01
BMB	0.67	0.00	0.67	3.29	22.25	208.20	25.05	57.24
ILS	0.67	0.00	0.67	2.19	21.79	150.40	23.80	49.37
ILS-ES	0.67	0.00	0.67	17.53	45.39	3512.00	92.50	12.85
ILS-ES Adapt.	0.67	0.00	0.67	4.29	22.19	166.40	24.43	37.44
COPKM	0.67	6.20	0.69	0.02	37.31	222.80	40.30	0.30
BCA	0.68	19.40	0.75	27.98	38.39	2397.20	70.55	51.95
RCA	0.67	0.00	0.67	8.18	21.90	144.80	23.84	180.66

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
BL	0.72	0.00	0.72	0.38	13.59	52.00	14.55	1.03
ES	0.72	0.00	0.72	2.27	12.92	100.60	14.77	15.60
BMB	0.72	0.00	0.72	3.20	14.29	0.00	14.29	6.81
ILS	0.72	0.00	0.72	1.68	12.92	92.80	14.64	3.70
ILS-ES	0.72	0.00	0.72	16.77	12.98	2214.80	53.90	3.10
ILS-ES Adapt.	0.72	0.00	0.72	3.64	14.29	0.00	14.29	5.23
COPKM	0.72	0.00	0.72	0.02	14.51	170.20	17.66	0.03
BCA	0.74	16.80	0.81	27.82	14.18	246.40	18.73	31.32
RCA	0.72	0.00	0.72	8.78	13.60	47.00	14.47	14.72

Al analizar en detalle los números, las conclusiones resultantes son las siguientes:

Basic Competitive Adaptation (BCA) es suficiente, pero no particularmente robusto. A pesar de que eventualmente alcanza el óptimo en los conjuntos de datos sencillos, iris y rand, no cuenta con la consistencia necesaria para afirmar que se trata de una buena metaheurística. En particular, para los casos con 20% de restricciones, difícilmente llega al óptimo en estos conjuntos de datos. Al analizar este detalle, resulta evidente que la principal debilidad de este algoritmo se halla en la forma en la que maneja (o deja de manejar) las restricciones. Las desviaciones generales son bastante cercanas a los resultados de los otros algoritmos estudiados, pero las tasas de restricciones violadas sobrepasan por mucho al resto, siendo el único caso donde no es la infeasibility más alta en los conjuntos difíciles (newthyroid y ecoli), en el que ILS con Enfriamiento Simulado en su versión original da peores resultados, pero es necesario recordar que en ese caso la causa era una mala adaptación de los parámetros, y que al adaptarlos adecuadamente (ILS-ES adaptado), los resultados mejoran notablemente.

Este algoritmo, además, es con diferencia de los que más tiempo de ejecución requiere para los conjuntos fáciles, sin ningún tipo de mejora en tiempo para los conjuntos difíciles como contraposición. Las causas de esto es que el mecanismo poblacional requiere contemplar múltiples torneos con soluciones "malas" hasta ir, estocásticamente, encontrando mejores soluciones.

Refined Competitive Adaptation (RCA) hace valer su título de refinado. La hibridación memética hace que los resultados estén a la par de las técnicas estudiadas previamente, e incluso superando a varias de ellas para ciertos conjuntos. Como elemento distintivo, tiene casos para rand y newthyroid donde no llega al óptimo, posible señal de que los mecanismos, aunque poderosos, requieren trabajo para asegurar mayor consistencia de resultados. En ecoli, sin embargo, es donde enseña su poder: es el segundo mejor resultado para ambos porcentajes, tan solo marginalmente superado por ILS. En newthyroid la historia es similar, estando muy ligeramente por detrás de BMB y compitiendo en ese intervalo de 0,20 en la función objetivo con otras alternativas.

En cuanto a tiempo, si bien sigue siendo de las técnicas que más tiempo consume, la estrategia de convergencia alternativa del metajuego resuelto muestra su poder para la mayoría de conjuntos de datos, donde recorta hasta a un tercio del tiempo que requería originalmente BCA, quien

no cuenta con dicha optimización. Aún así, es de los algoritmos más lentos para los conjuntos sencillos, que lo hace posicionarse como una mala opción ante estos. En el caso de ecoli, donde sus resultados son notablemente buenos, lo paga con el tiempo requerido más alto de todos los algoritmos para un conjunto de datos dado, llegando a requerir más de 3 minutos.

9. Posibles optimizaciones y mejoras

El algoritmo de adaptación competitiva básico sienta las bases para el estudio de la calidad relativa de las soluciones como motor principal para balancear la explotación contra la exploración, que en su versión refinada muestra claramente que es una estrategia poderosa y con mucha capacidad de mejora, pero al momento de la implementación inicial, los resultados claramente son desfavorables, debido a su uso excesivo de elementos estocásticos como principal mecanismo de exploración y explotación. Por tanto, en ese aspecto es dudoso que existan mejoras sustanciales que no involucren alterar trozos significativos del algoritmo, tal y como hace su versión refinada (RCA).

Sobre esta última, el interés es mucho más evidente y notable. El algoritmo en su primera modificación ya plantea resultados competitivos en los conjuntos de datos más complejos. Sus principales deficiencias son la consistencia y la velocidad. El hecho de que para conjuntos sencillos no alcance consistentemente el óptimo es síntoma que podría estar explotando mucho más de lo que explora, estancándose en ciertas ejecuciones en óptimos locales. Por otro lado, la mejora de metajuego resuelto es efectiva al momento de converger en conjuntos sencillos, pero con conjuntos complejos como ecoli donde no se alcanza esta condición, queda evidenciado que el mecanismo de búsqueda local, aún con las abstracciones y corta cantidad de iteraciones, sigue demandando demasiados recursos para utilizarse con tanta frecuencia.

La principal mejora que puede plantearse es que los porcentajes de explotación de cada puesto del podio varíen en vez de ser constante, de forma tal que aunque siempre se explote el primer lugar más que el segundo, no necesariamente sean en la misma proporción para cada torneo. De esta forma, damos espacio a mayor exploración en vez de explotar soluciones que aparenten ser efectivas a primera impresión. Una forma de hacer esto es a través de la mejora percibida entre torneos: si la estrategia que está en primer lugar apenas mejora, mientras que la que está en segundo lugar mejora notablemente, tendría sentido explotar más la opción en segundo lugar, pues resulta más prometedora la posibilidad de encontrar una mejora, mientras que aquella en primer lugar parece haber alcanzado un óptimo local y se harán un montón de adaptaciones que finalmente convergerán en

la misma solución. La implementación de esta idea se consideró pero por razones de tiempo no se pudo llevar a cabo. Sin embargo, durante su diseño se hizo notar que incluso, para mejorar aún más la exploración respecto a la explotación de soluciones óptimas locales, podría eliminarse una solución del podio si se considera que no hay más mejora posible, pero en este caso es necesario reconsiderar el mecanismo de elitismo de la implementación original eliminado para la versión refinada, así como un histórico de los resultados de la estrategia en torneos anteriores, para así poder contabilizar su mejora o falta de. Además, este mecanismo dificulta enormemente la aplicación del mecanismo de metajuego resuelto, por lo que nuevamente los tiempos de convergencia, de por sí notablemente largos en ciertos casos, podrían llegar a extenderse mucho más.

10. Bibliografía

https://numpy.org/doc/1.17/

Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local. Óscar Cordón García.

Seminario 3. Problemas de optimización con técnicas basadas en poblaciones. Óscar Cordón García

Seminario 4. Técnicas basadas en trayectorias para el Problema de la Máxima Diversidad (MDP) y el Problema del Agrupamiento con Restricciones (PAR). Óscar Cordón García.

Material de teoría para el curso de Metaheurísticas 2019/2020. Francisco Herrera.

https://docs.python.org/3/library/pathlib.html

Constrained K-means Clustering with Background Knowledge. Proceedings of the Eighteenth International Conference on Machine Learning, 2001, p. 577–584. Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schroedl.

https://realpython.com/python-timer/

http://zetcode.com/python/fstring/

11. Agradecimientos

Quiero, por último, agradecer a los profesores de la asignatura por compartir su interés, ánimo y deseo de conocer la asignatura con nosotros el estudiantado, así como por su capacidad de comprensión ante la difícil situación bajo la que tocó llevar a cabo el cuatrimestre. Mis mejores deseos para ustedes de cara al futuro, y gracias por abrir las puertas a proyectos sumamente creativos e interesantes como éste.