UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

Escuela Técnica Superior de Ingenieros Informáticos

Asignatura: Seminarios

Informe
S7 (Computación Natural)

Pedro Larrañaga, Alfonso Rodríguez-Patón y Alfonso Mateos

Kevin Oscar Arce Vera (kevin.avera@alumnos.upm.es)

MUIA - Máster Universitario en Inteligencia Artificial

Mayo 2025

Índice

1. Introducción

1.1. Algoritmos de Estimación de Distribución (EDAs)

Los Algoritmos de Estimación de Distribución (EDAs, por sus siglas en inglés, *Estimation of Distribution Algorithms*) constituyen una clase de algoritmos de computación evolutiva que se distinguen por reemplazar los tradicionales operadores genéticos (como el cruce o la mutación) por el aprendizaje y el muestreo de un modelo probabilístico. La idea fundamental detrás de los EDAs es capturar la estructura de dependencia entre las variables que caracterizan las soluciones de alta calidad encontradas hasta el momento y utilizar esta información para generar nuevas soluciones prometedoras.

El ciclo básico de un EDA sigue una estructura iterativa similar a otros algoritmos evolutivos:

1. **Inicialización:** Se genera una población inicial de individuos de forma aleatoria o mediante un método específico.
2. **Evaluación:** Cada individuo de la población es evaluado mediante una función de coste (o de aptitud) que mide la calidad de la solución que representa.
3. **Selección:** Se seleccionan los mejores individuos de la población basándose en su evaluación. La proporción de individuos seleccionados está controlada típicamente por un parámetro (α).
4. **Aprendizaje del Modelo Probabilístico:** Se aprende un modelo probabilístico a partir de los individuos seleccionados. Este modelo busca representar la distribución de probabilidad de las soluciones de alta calidad.
5. **Muestreo:** Se genera una nueva población muestreando del modelo probabilístico aprendido.
6. **Condición de parada:** Si se cumple un criterio de parada (por ejemplo, número máximo de iteraciones alcanzado, mejora insuficiente, etc.), el algoritmo finaliza; de lo contrario, se regresa al paso 2.

La principal diferencia entre las distintas variantes de EDAs reside en el tipo de modelo probabilístico utilizado y cómo se aprende este modelo. La elección del modelo depende fundamentalmente de la naturaleza de las variables del problema a optimizar:

- **Variables Binarias o Categóricas:** Se utilizan modelos de probabilidad discreta. Las técnicas varían desde aprender las distribuciones marginales de cada variable de forma independiente (como en *Univariate Marginal Distribution Algorithm*, UMDA binario (Mühlenbein and Paass (1996)) o UMDA categórico) hasta modelos que capturan dependencias complejas entre las variables, a menudo representados mediante Redes Bayesianas (como en *Estimation of Bayesian Network Algorithm*, EBNA o *Bayesian Optimization Algorithm*, BOA).
- **Variables Continuas:** Se emplean modelos de probabilidad continua. Las técnicas incluyen:
  - Aprendizaje de distribuciones marginales univariantes, típicamente Gaussianas (como en UMDA continuo, UMDAc) o utilizando Estimación de Densidad Kernel (KDE) (como en *Univariate KEDA*). Estos modelos asumen independencia entre las variables.
  - Aprendizaje de modelos multivariantes que capturan dependencias entre las variables. Ejemplos incluyen la estimación de una distribución Gaussiana multivariante (como en *Estimation of Multivariate Normal Algorithm*, EMNA) o la construcción de Redes Bayesianas Gaussianas (como en *Estimation of Gaussian Distribution Algorithm*, EGNA).

1.2. Travelling Salesman Problem (TSP)

Este es uno de los problemas de optimización combinatoria más clásicos y estudiados en el ámbito de la informática y la investigación operativa. Su formulación es simple pero su resolución exacta para instancias grandes es computacionalmente muy costosa.

El problema se define a partir de un conjunto de N ciudades y una matriz de distancias o costes que especifica la "distancia"(que puede ser tiempo, coste, etc.) entre cada par de ciudades. El objetivo es encontrar la ruta o ciclo que visite cada ciudad exactamente una vez y regrese a la ciudad de origen, minimizando la distancia total recorrida. Formalmente, si las ciudades están etiquetadas de 1 a N, la solución es una permutación $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ de las ciudades $(1, 2, \ldots, N)$ tal que la suma de las distancias $d(\sigma_1, \sigma_2) + d(\sigma_2, \sigma_3) + \ldots + d(\sigma_{N-1}, \sigma_N) + d(\sigma_N, \sigma_1)$ es mínima.

Una característica fundamental del TSP es que pertenece a la clase de problemas NP-duros. Esto implica que no existe (hasta donde se sabe) un algoritmo determinista que pueda encontrar la solución óptima en tiempo polinomial con respecto al número de ciudades. A medida que el número de ciudades aumenta, el espacio de búsqueda (el número de posibles rutas, que es $(N-1)!/2$ para TSP simétrico) crece de forma factorial, volviendo inviable la búsqueda exhaustiva incluso para valores moderados de N.

Existen diversas variantes del TSP, siendo una de las más comunes el TSP Euclidiano, donde las ciudades son puntos en un plano (como en el problema a resolver en este trabajo) y las distancias entre ellas se calculan utilizando la métrica Euclidiana estándar. En este caso, el problema es siempre simétrico ($d(i, j) = d(j, i)$).

Debido a su dificultad computacional y su naturaleza representativa de una amplia gama de problemas de planificación y secuenciación en el mundo real (logística, ruteo de vehículos, perforación de placas de circuitos impresos, etc.), el TSP se ha convertido en un problema de referencia utilizado para evaluar el rendimiento de algoritmos de optimización, incluyendo algoritmos exactos (que solo son prácticos para instancias pequeñas), heurísticas y metaheurísticas como los algoritmos evolutivos y, en particular, los EDAs.

1.3. EDAspy

EDAspy es una librería de código abierto escrita en Python diseñada para facilitar la implementación y experimentación con los Algoritmos de Estimación de Distribución (EDAs). Su principal cometido es proporcionar un *framework* modular y extensible que permita a investigadores y desarrolladores aplicar diversas variantes de EDAs a problemas de optimización de forma sencilla. Si bien la librería tiene un enfoque particular en la optimización continua, incluye implementaciones para variables binarias y categóricas, y ofrece herramientas para la construcción de EDAs personalizados (Soloviev et al. (2024)).

La librería encapsula los pasos fundamentales de un EDA (inicialización, selección, aprendizaje del modelo, muestreo) y abstrae las particularidades de los diferentes modelos probabilísticos, permitiendo al usuario centrarse en la definición de la función de coste del problema a resolver y la configuración de los parámetros del algoritmo.

EDAspy incluye una variedad de implementaciones de EDAs, cubriendo diferentes tipos de variables y niveles de complejidad en el modelado de dependencias:

- **UMDAd (Univariate Marginal Distribution Algorithm binary):** EDA básico para variables binarias. Modela la distribución de cada variable de forma independiente.
- **UMDAc (Univariate Marginal Distribution Algorithm continuous):** Variante para variables con-

tinuas. Modela cada variable de forma independiente asumiendo una distribución Gaussiana (Normal). Es particularmente útil para problemas como la optimización de hiperparámetros.

- **UnivariateKEDA (Univariate Kernel Estimation of Distribution Algorithm):** Para variables continuas. Estima la distribución marginal de cada variable utilizando Estimación de Densidad Kernel (KDE), ofreciendo más flexibilidad que la asunción Gaussiana.
- **UMDAcat (Univariate Marginal Distribution Algorithm categorical):** Versión para variables categóricas con más de dos posibles valores. Modela cada variable de forma independiente.
- **EGNA (Estimation of Gaussian Distribution Algorithm):** EDA avanzado para variables continuas. Aprende y muestrea a partir de una Red Bayesiana Gaussiana, permitiendo capturar y explotar dependencias entre las variables.
- **EMNA (Estimation of Multivariate Normal Algorithm):** Similar a EGNA para variables continuas, pero utiliza una distribución Gaussiana multivariante (basada en la matriz de covarianza) para modelar las dependencias.
- **SPEDA (Semiparametric Estimation of Distribution Algorithm):** Un enfoque multivariante basado en archivo que permite modelar variables continuas utilizando una combinación de KDE y Gaussianas, capturando dependencias.
- **MultivariateKEDA:** Caso especial de SPEDA donde todas las variables continuas son modeladas utilizando KDE.
- **EBNA (Estimation of Bayesian Network Algorithm):** Para variables categóricas. Aprende y muestrea a partir de una Red Bayesiana Categórica para modelar dependencias.
- **BOA (Bayesian Optimization Algorithm):** Similar a EBNA para variables categóricas, pero utiliza un criterio de puntuación específico (Bayesian Dirichlet score) para el aprendizaje de la estructura de la Red Bayesiana.
- **PBIL (Population-based Incremental Learning):** Una modificación de la estrategia UMDA que ajusta la media de las distribuciones marginales considerando no solo los mejores individuos, sino también los peores.

La diversidad de implementaciones disponibles en EDAspy permite abordar una amplia gama de problemas de optimización. En el contexto de este trabajo, EDAspy será la herramienta principal utilizada para aplicar algoritmos de estimación de distribución a la resolución del TSP.

2. Implementación

2.1. Virtualización

Durante el desarrollo de esta parte práctica se ha empleado un entorno virtualizado como el que se puede crear con herramientas como Anaconda. De esta forma se facilita la relicabilidad de los resultados obtenidos. Junto a este documento se adjuntará el archivo *environment.yml* que define el entorno utilizado.

2.2. Librería tsp95

Para medir de una forma más estandarizada los resultados que lograban los EDAs hemos empleado la librería *tsp95lib*(Reinelt (1991)). Tiene una colección de instancias del problema TSP, en muchos casos con un coste óptimo encontrado, y con una variedad de complejidad del problema amplia (desde 70 hasta 2392 ciudades).

Concretamente se probará el algoritmo en las siguientes instancias aportadas por la librería:

Cuadro 1: Costos óptimos y tamaños de instancias seleccionadas del TSP (TSPLIB).

problem_name	**optimal_cost**	**size**
st70	675	70
pr76	108159	76
rd100	7910	100
lin105	14379	105
tsp225	3916	225
pcb442	50778	442
pr1002	259045	1002
pr2392	378032	2392

2.3. Representación del problema

La aplicación de algoritmos de optimización, incluyendo los EDAs, a problemas como el TSP requiere definir una forma de *codificar* las posibles soluciones (las rutas) de manera que sea compatible con el tipo de variables que el algoritmo maneja. Dado que la mayoría de las implementaciones en la librería EDAspy están orientadas a la optimización continua, se opta por una representación que transforme la naturaleza permutacional del TSP a un dominio continuo.

La siguiente opción que surge es codificar el problema con un vector que represente la ordenación de las ciudades de otra forma. En concreto, se utiliza un vector de **valores continuos** de tamaño N, donde N es el número total de ciudades en la instancia del TSP. Este vector se denota como $\mathbf{v} = [v_0, v_1, \ldots, v_{N-1}]$.

Cada posición i en este vector continuo ($0 \leq i < N$) se asocia unívocamente con una de las ciudades del problema (utilizando el índice interno de la ciudad, por ejemplo). El valor continuo v_i en la posición i no representa la ciudad en sí ni su posición en la ruta directamente, sino una especie de "prioridad.º "preferencia.ªsignada a la ciudad asociada a esa posición.

Para transformar este vector continuo $\mathbf{v}$ en una solución válida para el TSP (es decir, una permutación de las ciudades), se aplica un proceso de **decodificación basado en la ordenación**:

1. Se crea una lista de pares (valor_continuo, índice_ciudad), donde el índice_ciudad es el índice interno original (0 a $N-1$) correspondiente a la posición en el vector $\mathbf{v}$.
2. Se ordena esta lista de pares basándose en los valores continuos.
3. La permutación de ciudades (la ruta) se obtiene tomando los índices de ciudad en el orden en que aparecen después de la ordenación. La ciudad con el valor continuo más bajo en $\mathbf{v}$ será la primera en la ruta, la ciudad con el segundo valor más bajo será la segunda, y así sucesivamente.

Formalmente, si al ordenar $\mathbf{v}$ obtenemos $v_{i_0} < v_{i_1} < \ldots < v_{i_{N-1}}$, donde $i_0, i_1, \ldots, i_{N-1}$ son los índices originales de las variables en $\mathbf{v}$, la ruta decodificada será la secuencia de ciudades correspondiente a los índices internos $i_0, i_1, \ldots, i_{N-1}$.

La ventaja fundamental de esta codificación es que el proceso de ordenación *garantiza* que, si los valores continuos son distintos (lo cual es prácticamente siempre el caso al trabajar con números de punto flotante), el resultado será una **permutación válida** de los N índices de ciudad. Esto evita la necesidad de lidiar con soluciones inválidas (rutas que repiten u omiten ciudades) durante el proceso de optimización.

De esta forma, el problema original de optimización combinatoria del TSP se reformula como un problema de optimización continua: encontrar el vector $\mathbf{v} = [v_0, \ldots, v_{N-1}]$ que, una vez decodificado mediante ordenación en una permutación, minimice la longitud total de la ruta resultante. La función de coste que el EDA intenta minimizar recibe el vector continuo $\mathbf{v}$, lo decodifica a una permutación y calcula la distancia total del ciclo, incluyendo el retorno a la ciudad de inicio. Esta representación permite la aplicación de algoritmos de EDAs continuos, como UMDAc, EGNA o EMNA, disponibles en la librería EDAspy.

3. Resultados

3.1. Exploración de hiperparámetros

Para comprender el impacto de los parámetros del algoritmo en el rendimiento, comenzamos explorando el efecto de dos hiperparámetros clave de UMDAc: *size_gen* (tamaño de la población) y *alpha* (porcentaje de individuos seleccionados para el aprendizaje del modelo), utilizando la instancia del problema *st70*. El coste óptimo conocido para *st70* es de 675. Los resultados obtenidos para diversas combinaciones de *size_gen* y *alpha* se resumen visualmente en la Figura 1, que muestra un mapa de calor del coste final en una sola ejecución.

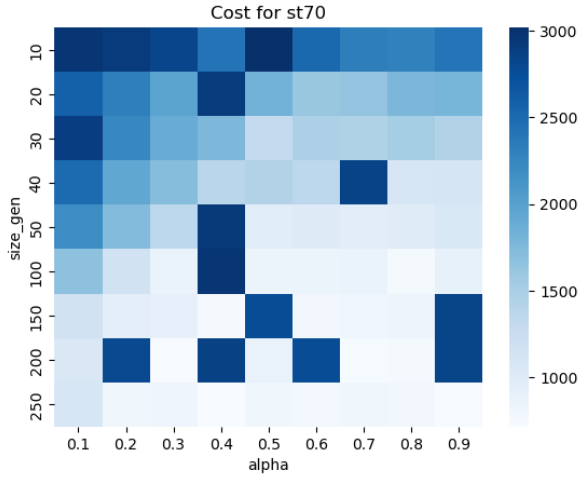

Figura 1: Efecto de *alpha* y *size_gen* en coste de UMDAc para TSP st70 (una ejecución)

Como se puede observar en la figura y en los datos subyacentes, el tamaño de la población (*size_gen*) tiene un impacto significativo y generalmente positivo en la calidad de la solución encontrada. Un valor insuficiente de *size_gen* (por ejemplo, 10 o 20) conduce a costes considerablemente más altos, muy alejados del óptimo. Esto se debe a que una población pequeña proporciona una muestra limitada del espacio de búsqueda. El modelo probabilístico aprendido a partir de un conjunto reducido de individuos tiende a ser menos representativo de la distribución de soluciones de alta calidad, lo que dificulta la exploración efectiva de las regiones prometedoras del espacio de búsqueda en iteraciones posteriores.

El parámetro *alpha*, que determina la presión selectiva al seleccionar el subconjunto de individuos para aprender el modelo, también influye notablemente en los resultados:

- Un valor de *alpha* demasiado pequeño (por ejemplo, 0.1 o 0.2), especialmente cuando se combina con un *size_gen* bajo, a menudo resulta en un rendimiento pobre. Al seleccionar solo una fracción muy pequeña de la población (la élite más extrema), el modelo probabilístico se basa en datos muy específicos que podrían no capturar la diversidad de características presentes en soluciones de buena calidad en general. Esto puede llevar a una convergencia prematura, donde el algoritmo se centra rápidamente en una región subóptima del espacio de búsqueda y no logra escapar de ella.
- A medida que *size_gen* aumenta, el impacto negativo de un *alpha* pequeño disminuye, pero valores intermedios a altos de *alpha* (aproximadamente entre 0.4 y 0.8 en este experimento) parecen ofrecer consistentemente los mejores resultados, acercándose más al coste óptimo conocido. Esto sugiere que, para el problema *st70* y utilizando esta representación, un equilibrio entre una presión selectiva moderada/alta

(controlada por *alpha*) y una muestra amplia para aprender el modelo (controlada por *size_gen*) es un aspecto a considerar.

Es relevante notar que incluso con combinaciones de hiperparámetros que generalmente producen buenos resultados (como *size_gen* altos y *alpha* intermedios), pueden aparecer ocasionalmente resultados subóptimos (visibles como puntos más claros en el mapa de calor dentro de las regiones oscuras). Esto refleja la naturaleza estocástica de los algoritmos evolutivos y de los EDAs en particular; una única ejecución podría no encontrar siempre la mejor solución posible dentro del número máximo de iteraciones, incluso con parámetros adecuados.

3.2. Comparación con otros EDAs

Como UMDAc no modela dependencias probamos con EMNA, que si lo hace y no tiene tanta carga computacional al aprender distribuciones multivariantes Gausianas (en comparación con EGNA que usa Redes Bayesianas Gausianas). Al igual que con *UMDAc* exploramos los hiperparámetros *size_gen* y *alpha* en distintas combinaciones. En esta ocasión nos quedaremos con el mejor resultado de tres ejecuciones por combinación. Debido a problemas de convergencia de *EMNA*, restringimos el espacio de exploración para ambos hiperparámetros. Y para hacer una comparación justa con *UMDAc* repetiremos el experimento en este EDA.

Los resultados son los mostrados en las imágenes 2 y 3

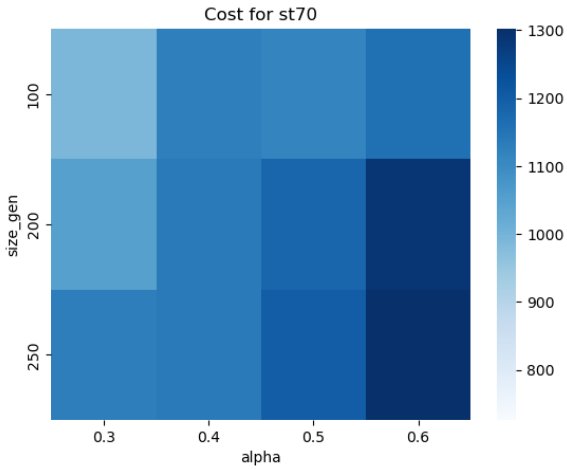

Figura 2: Coste de EMNA para TSP st70 (best of 3)

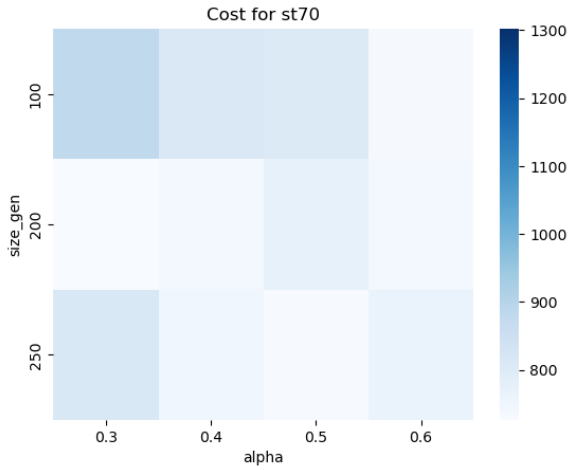

Figura 3: Coste de UMDAc para TSP st70 (best of 3)

En la totalidad de casos se observa una mejor solución con el EDA *UMDAc*. Esto es contrario a lo que esperaba debido a que UMDAc asume independencia entre variables. Esto no se debe a que las soluciones de *EMNA* se hayan logrado tras alcanzar el máximo de iteraciones establecidas, en la mayoría de casos las soluciones se consiguen debido a que el EDA se estanca. Quedándose atrapado en soluciones de las que no se consigue salir.

4. Conclusiones

Este trabajo práctico ha permitido aplicar y explorar los Algoritmos de Estimación de Distribución (EDAs) proporcionados por la librería EDAspy a la resolución del Problema del Viajante de Comercio (TSP), un problema clásico de optimización combinatoria.

Los principales aprendizajes y resultados obtenidos son los siguientes:

- Se ha validado la viabilidad de utilizar una **representación basada en un vector continuo** y un proceso de **decodificación por ordenación** para adaptar la estructura permutacional del TSP a algoritmos de optimización continua como los disponibles en EDAspy. Esta aproximación garantiza la generación de soluciones válidas (permutaciones completas).
- La **exploración de hiperparámetros** en el algoritmo UMDAc para la instancia *st70* demostró la importancia crítica del **tamaño de la población (*size_gen*)** y el parámetro de **selección (*alpha*)**. Un tamaño de población suficientemente grande es esencial para que el modelo probabilístico capture adecuadamente la distribución de soluciones prometedoras. El parámetro *alpha* influye en la presión selectiva, observándose que valores intermedios a altos tienden a arrojar mejores resultados al encontrar un balance entre explotación de las mejores soluciones y exploración del espacio de búsqueda. Los resultados confirmaron la naturaleza estocástica de estos algoritmos y la necesidad de sintonización para cada problema.
- La **comparativa entre UMDAc (univariante)** y **EMNA (multivariante)**, ambos aplicados con la representación continua, arrojó un resultado inesperado. Contrario a la expectativa teórica de que EMNA, al modelar dependencias entre variables, podría ser más adecuado para la estructura implícita de la representación, UMDAc consistentemente obtuvo soluciones de mejor calidad en el espacio de parámetros explorado. Se observó que EMNA tendía a estancarse prematuramente en óptimos locales para esta instancia particular. Esto sugiere que, aunque los modelos multivariantes tienen un mayor potencial teórico, su rendimiento práctico puede depender significativamente de la instancia del problema, la representación utilizada y la robustez de la implementación específica del algoritmo.

En suma, la experiencia con EDAspy para resolver el TSP mediante una codificación continua y el análisis de sus resultados refuerza la comprensión de los mecanismos de los EDAs, la relevancia de la representación del problema y la necesidad de experimentación empírica para validar el comportamiento de los algoritmos en instancias específicas.

Referencias

Mühlenbein, H. and Paass, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In *International conference on parallel problem solving from nature*, pages 178–187. Springer.

Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384.

Soloviev, V. P., Larrañaga, P., and Bielza, C. (2024). Edaspy: An extensible python package for estimation of distribution algorithms. *Neurocomputing*, 598:128043.