Evaluación de Distintas Bibliotecas de Optimización para Python

Baglietto Fernandez Agustín
abaglietto@fi.uba.ar
Departamento de Gestión, Facultad de Ingeniería
Universidad de Buenos Aires

Resumen

La programación lineal es una técnica matemática clave para resolver problemas de optimización con restricciones, aplicable en logística, producción y transporte entre otros. Este trabajo evalúa diferentes bibliotecas de Python como PuLP, OR-Tools, Pyomo, PySCIPOpt, CVXPY y CPLEX, enfocándose en su rendimiento al resolver el problema del viajante de comercio (o TSP, por sus siglas en inglés, *Traveling Salesman Problem*). Se analizan tiempos de ejecución, escalabilidad y algoritmos, utilizando Python 3.9.0 en Visual Studio Code. Los resultados demostraron que bibliotecas de código abierto como HIGHS y SCIP son eficientes para problemas pequeños y medianos, mientras que bibliotecas comerciales como CPLEX y Gurobi destacan en problemas de mayor envergadura. El estudio ofrece una guía práctica para seleccionar herramientas de optimización según la complejidad del problema.

1. Introducción

La programación lineal [11] es un método matemático para la optimización de problemas o modelos con restricciones. Estos problemas poseen un funcional o función objetivo que consiste en una ecuación lineal en donde se busca optimizar, mediante minimización o maximización, el resultado de la misma. Además, estos problemas cuentan con restricciones que se expresan mediante inecuaciones lineales de mayor o igual, menor o igual o, directamente, igualdades. Algunos ejemplos de estos problemas orientados a la ingeniería industrial son la optimización de la producción, los procesos, asignación de eficiencia de recursos, distribución, transporte, ruteo, planificación de turnos, asignación de personal, logística, gestión de inventarios, *blend* de productos, toma de decisiones y control de calidad, entre otros.

Dependiendo del tipo de variables que integran estos problemas podemos clasificarlos como:

- Programación Lineal (LP, por sus siglas en inglés, *Linear Programming*): donde las variables utilizadas son continuas.
- Programación Lineal Entera Mixta (MILP, por sus siglas en inglés, *Mixed Integer Linear Programming*): donde algunas de las variables de decisión que integran de modelo son continuas mientras que otras son enteras.
- Programación Lineal Entera (IP, por sus siglas en inglés, *Integer Programming*): donde todas las variables de decisión son de tipo entero.

Este trabajo se enfoca en la resolución de un problema de tipo MILP relacionado con la distribución y transporte.

El aumento en la capacidad de procesamiento computacional permite abordar modelos de mayor complejidad y magnitud. Las bibliotecas y los distintos algoritmos de los motores de optimización en Python, o cualquier otro lenguaje de programación, son fundamentales para la resolución rápida de diversos problemas dentro de escenarios complejos.

2. Problema del viajante de comercio (TSP)

El modelo a resolver para comparar las distintas bibliotecas y algoritmos de resolución es el del viajante de comercio [1]. Este problema simula a un comerciante que debe viajar por una determinada cantidad de ciudades para comercializar su producto en cada una, minimizando la distancia recorrida y visitando una sola vez cada lugar. Para llevar a cabo este recorrido, se debe contar con un solo camino evitando así los subtours.

Las ecuaciones planteadas para este problema son:

2.1. Restricciones

- **Restricción de solo una entrada a cada nodo:**

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall\, j \neq i \tag{1}$$

- **Restricción de solo una salida de cada nodo:**

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall\, i \neq j \tag{2}$$

- **Restricción de subtours (Miller-Tucker-Zemlin [1]):**

$$u_i - u_j + (n-1)x_{ij} \leq n-2, \quad \forall\ i,\, j, i \neq j,\, i,\, j = 2, ..., n \tag{3}$$

2.2. Funcional

$$\text{mín } Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{4}$$

donde:

- x_{ij}: variable binaria, donde si existe el recorrido del nodo i al nodo j, toma valor 1, sino 0.
- d_{ij}: distancia del nodo i al nodo j.
- u_i: variable de posición del nodo i, para restricción de subtours.

Este problema se encuadra dentro de la programación lineal, más precisamente optimización combinatoria. Esto es debido a que para n ciudades el número posible de soluciones es $(n-1)!$, es decir, la cantidad de soluciones posibles crece de forma factorial con el número de nodos.

El problema del viajante de comercio es considerado un problema *NP-Hard* [1], ya que el esfuerzo computacional necesario para cualquier método de resolución crece de forma no polinomial con el tamaño del problema, es decir, no existe un algoritmo polinomial para el TSP .

En este caso, se plantea la resolución para $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$.

2.3. Cantidad de posibles soluciones

Número de nodos	**N° de posibles soluciones**
10	362.880
20	$1{,}216 \times 10^{17}$
30	$8{,}841 \times 10^{27}$
40	$2{,}039 \times 10^{47}$
50	$3{,}041 \times 10^{63}$
60	$4{,}278 \times 10^{78}$
70	$1{,}197 \times 10^{94}$
80	$7{,}156 \times 10^{110}$
90	$8{,}659 \times 10^{126}$
100	$9{,}332 \times 10^{155}$

Tabla 1: cantidad de soluciones por número de nodos del problema.

3. Bibliotecas de Python

Una biblioteca de Python es una colección de código programado por desarrolladores la cual posee módulos y funciones para realizar tareas específicas sin la necesidad de programarlas uno mismo para poder utilizarlas. Python posee bibliotecas internas que se incluyen al instalar el lenguaje, y existen otras

que deben ser instaladas externamente mediante el administrador de paquetes de Python. Existen bibliotecas para diferentes y variadas utilidades, es así que cualquier programador puede desarrollar su propia biblioteca. En este trabajo se analizan aquellas que tienen como finalidad la optimización matemática y resolución de problemas de programación lineal. Para resolver el problema planteado de programación lineal entera mixta se pueden utilizar distintas bibliotecas, las analizadas en este trabajo se presentan en la Tabla 2.

Biblioteca	**Creador**	**Año**	**Código abierto**	**Licencia**
PuLP	John D. Hunter	2006	Sí	GNU LGPL
OR-Tools	Google	2016	Sí	Apache 2.0
Pyomo	Sandia NL	2008	Sí	BSD-3-Clause
PySCIPOpt	ZIB	2016	Sí	MIT
CVXPY	Steven Diamond	2014	Sí	Apache 2.0
CPLEX	IBM	2010	No	Comercial

Tabla 2: bibliotecas analizadas en el estudio.

- **PuLP:** es un modelador de programación lineal y programación lineal entera mixta escrito en Python [2]. Con PuLP, es fácil crear problemas de optimización y resolverlos utilizando los algoritmos de código abierto más recientes o más importantes.
- **OR TOOLS:** Google Optimization Tools (OR-Tools) es un software de código abierto, rápido y portable para la resolución de problemas de optimización combinatoria con múltiples y variados motores de resolución [4].
- **Pyomo:** es una librería basada en Python de código abierto que soporta distintos algoritmos de optimización para formular, resolver y analizar problemas de optimización [5].
- **PySCIPOpt:** este proyecto provee de una interface para Python del sistema de resolución SCIP [6] y no permite utilizar otros.
- **CVXPY:** es un lenguaje de modelado de código abierto integrado en Python para problemas de optimización convexa. Permite expresar los problemas de manera natural, siguiendo la notación matemática en lugar de la forma restrictiva que requieren los algoritmos de resolución [7].
- **CPLEX:** interfaz de Python proporcionada por IBM para interactuar con su algoritmo de optimización CPLEX. Permite modelar y resolver problemas de optimización matemática (LP, MILP, programación cuadrática) desde Python. Cabe aclarar que Cplex fue desarrollada en 1988 por Robert E. Bixby, luego en 1997 fue adquirida por ILOG, para en 2009 ser comprada por IBM, quien la continúa desarrollando.

4. Algoritmos de resolución

Existen diferentes algoritmos de resolución los cuales pueden ser utilizados con las diversas bibliotecas de Python mencionadas anteriormente. En este trabajo se analizan ocho de los más importantes en la resolución del problema TSP, se presentan en la tabla 2. Algunos de estos motores son capaces de resolver una gran variedad de tipos de problemas, mientras que otros están diseñados para abordar problemas específicos. En la Tabla 3 se listan los algoritmos utilizados, junto con sus características.

Algoritmo de Resolución	**Creador**	**Año**	**Código Abierto**	**Licencia**
CBC	COIN-OR	2002	Sí	Eclipse Public Lic.
Gurobi	Gurobi Inc.	2008	No	Comercial
HiGHS	Univ. Edimburgo	2017	Sí	MIT
CPLEX	IBM	1988	No	Comercial
XPRESS	Dash Opt.	1983	No	Comercial
SCIP	ZIB	2004	Sí	Apache 2.0
SAT	Google	2016	Sí	Apache 2.0
GLPK	GNU	2001	Sí	GPL v3

Tabla 3: algoritmos de resolución analizados en el estudio.

La mayoría de estos motores de resolución utilizan diversos algoritmos de optimización según el tipo de problema a resolver. Gurobi selecciona automáticamente el algoritmo más adecuado en función de las características del problema y puede cambiar entre ellos durante la ejecución en busca de mayor eficiencia. Tiene licencia de pago para problemas grandes y comerciales, es un motor de alto rendimiento.

Cplex tiene una limitación de problemas con contenido de hasta mil variables en su licencia gratuita, luego para problemas más grandes y comerciales requiere una licencia de pago. Es uno de los algoritmos de resolución más potentes.

XPRESS es propiedad del grupo FICO y requiere licencia de pago para problemas grandes.

Los motores de resolución gratuitos analizados son HiGHS, SCIP (para proyectos comerciales requiere pago), Glpk, CBC. SAT es gratuito pero no es adecuado para resolver problemas lineales. En cambio, es útil para resolver problemas de optimización combinatoria como es el TSP, al convertirlo en otro formato.

Para resolver los problemas de programación lineal, todos los motores de resolución utilizan los algoritmos Simplex [11, pp. 37-49] y de puntos interiores. CPLEX se distingue ya que también emplea el método Dual Simplex [11, pp. 81-83] y, junto con XPRESS, utiliza la barrera primal-dual.

Con respecto a los problemas de programación lineal entera mixta, todos los motores de resolución utilizan los algoritmos *branch and bound* [11, pp. 160-167], *branch and cut* [12] y heurísticas [14]. El único que no emplea heurísticas

es CBC. CPLEX utiliza relajación lagrangiana [14] y de frontera, eliminando la restricción de variables exclusivamente enteras. XPRESS incorpora además búsqueda local, mejorando iterativamente una posible solución inicial mediante movimientos locales, sin necesidad de explorar todo el espacio de soluciones, y, además utiliza la descomposición de Dantzig-Wolfe [13].

5. Metodología

Para la implementación del código se utilizó Python 3.9.0 en el entorno de desarrollo Visual Studio Code. Se analizaron las distintas bibliotecas presentadas en la tabla 1 con diferentes optimizadores presentados en la tabla 2. El rendimiento de cada uno se evaluó resolviendo el problema TSP programado para cada biblioteca, tal como se presentó en la introducción al mismo. Se analizaron el tiempo de ejecución en segundos, la cantidad de variables y la distancia total del recorrido para corroborar que la resolución fue óptima.

Algunos de los algoritmos de optimización están limitados en cuanto a la cantidad de variables y restricciones debido a licencias. En estos casos, se procedió a calcular hasta la cantidad máxima de nodos permitidos, dejando un guion en los casos en los que no se pudo calcular. Otros algoritmos de optimización requieren tiempos elevados para la resolución a medida que se incrementa la cantidad de nodos. En estos casos, no se realizaron los cálculos, por lo que se insertó nuevamente un guion en la tabla de resultados. Otro de los aspectos que resulta importante destacar es que la medición de los tiempos de resolución varía de corrida en corrida, siempre manteniendose en el orden de los tiempos presentados en la próxima sección de resultados.

6. Resultados

Los tiempos de ejecución para cada biblioteca y algoritmo se presentan en tablas, permitiendo comparar su rendimiento según la cantidad de nodos en el problema.

Se presentan los resultados para la biblioteca PuLP:

nodos	CBC	Gurobi	HiGHS	Cplex	XPRESS	SCIP	distancia
10	0.16	0.05	0.05	0.05	0.01	0.02	155.67
20	4.39	0.26	0.27	0.27	0.04	0.09	248.85
30	3.16	0.48	0.43	0.42	0.8	0.91	310.16
40	22	0.29	0.3	0.42	4.18	6.03	360.35
44	286	0.93	0.89	7.3	1.84	-	375.48
50	87	-	5.51	1.43	17.22	3.73	390.6
60	284.1	-	7.87	4.51	-	48.43	469.46
70	3451	-	95.85	17.83	-	159	530.9
80	-	-	266.21	43	-	98	553.29
90	-	-	258.51	29.96	-	156	586.7
100	-	-	260.46	70.76	-	233.23	649.41

Tabla 4: tiempos de resolución del TSP por algoritmo para la biblioteca PuLP.

Se observa que hasta los 44 nodos, que es la limitación que poseen los optimizadores comerciales, los mejores tiempos de resolución los presentan estos últimos junto con HiGHS y SCIP que son gratuitos. La diferencia entre los comerciales y estos se percibe a partir de los 70 nodos.

Se presentan los resultados para la biblioteca OR Tools:

nodos	CBC	Gurobi	SCIP	SAT	HiGHS	distancia
10	0.29	0.01	0.06	0.03	0.01	155.67
20	6.38	0.31	1.07	0.42	0.81	248.85
30	7.42	0.16	1.25	0.78	0.94	310.16
40	17.4	0.35	2.92	0.32	0.3	360.35
44	299.77	1.38	17.5	6.73	4.17	375.48
50	260.15	-	6.13	7.02	10.58	390.6
60	384.21	-	70	17.53	23.92	469.46
70	-	368.11	59.29	80.91	95	530.9
80	-	707.49	130.93	101.33	101.33	553.29
90	-	699.24	198.63	169.93	169.93	586.7
100	-	825.24	482.44	172.67	172.67	649.41

Tabla 5: tiempos de resolución del TSP por algoritmo para la biblioteca OR Tools.

En OR Tools, Cplex queda limitado a las 1000 variables debido a la licencia gratuita por lo que a gran cantidad de nodos no es posible compararlo. Se observa que HiGHS nuevamente, para ser un optimizador libre y de código abierto, resuelve el problema en menor tiempo.

Se presentan los resultados obtenidos con la biblioteca PYOMO:

nodos	Gurobi	Glpk	Cplex	XPRESS	distancia
10	0.95	0.11	0.16	0.26	155.67
20	0.44	1.03	0.21	0.64	248.85
30	0.33	0.71	0.21	1.33	310.16
40	0.98	27.45	0.96	6.29	360.35
44	1.73	170.84	1.94	10.42	375.48
50	-	3208	1.84	21.25	390.6
60	-	-	3.03	-	469.46
70	-	-	9.25	-	530.9
80	-	-	15.49	-	553.29
90	-	-	92.39	-	586.7
100	-	-	30.74	-	649.41

Tabla 6: tiempos de resolución del TSP por algoritmo para la biblioteca Pyomo.

En este caso se agregó el optimizador Glpk, pero se observa que no tiene un buen rendimiento a partir de los 40 nodos. En este caso las mejores opciones hasta los 44 nodos son Cplex y Gurobi, los dos comerciales, luego el único optimizador que permite calcular hasta los 100 nodos es Cplex, presentando unos tiempos de resolución muy bajos en comparación con otros algoritmos en otras bibliotecas.

Se presentan los resultados obtenidos con la biblioteca PySCIPOpt:

nodos	SCIP	distancia
10	0,04	155,67
20	0,66	248,85
30	0,28	310,16
40	0,45	360,35
50	2,75	390,6
60	18,55	469,46
70	93,96	530,9
80	217,45	553,29
90	161,89	586,7
100	308,67	649,41

Tabla 7: tiempos de resolución del TSP por algoritmos para la biblioteca PySCIPOpt.

PySCIPOpt solo permite resolver con su optimizador SCIP y en los resultados se observa que los tiempos de resolución se encuentran en el orden de el mismo utilizado en otras bibliotecas, por lo tanto no se distingue y posee menor versatilidad.

Se presentan los resultados obtenidos con la biblioteca CVXPY:

nodos	XPRESS	Cplex	HiGHS	SCIP	distancia
10	0,28	0,12	0,16	0,29	155,67
20	0,77	0,52	3,63	4,1	248,85
30	1,3	0,87	1,85	5,52	310,16
40	5,17	-	2,65	9,33	360,35
50	-	-	139,08	88,52	390,6
60	-	-	-	241,61	469,46
70	-	-	-	8161,91	530,9
80	-	-	-	16682,7	553,29
90	-	-	-	-	586,7
100	-	-	-	-	649,41

Tabla 8: tiempos de resolución del TSP por algoritmo para la biblioteca CVXPY.

Dado que CVXPY se encuentra diseñada para resolver principalmente problemas convexos, con los optimizadores que resuelven programación lineal y programación lineal entera mixta, a partir de los 40 nodos, presenta tiempos de resolución muy superiores a otras bibliotecas. Por esto no es una biblioteca óptima para este tipo de problemas.

Por último se presentan los resultados obtenidos en la biblioteca Cplex de IBM:

nodos	Cplex	distancia
10	0,1	155,67
20	0,18	248,85
30	0,31	310,16
40	-	-

Tabla 9: tiempos de resolución del TSP por algoritmo para la biblioteca Cplex.

Esta biblioteca sólo permite utilizar el optimizador Cplex y está limitada por licencia gratuita a 1000 variables. Para poder seguir resolviendo se debe comprar la licencia.

7. Conclusiones

La elección de la biblioteca y el algoritmo de resolución más adecuado depende de la magnitud del problema lineal a resolver, así como de si se trata de un proyecto comercial o de código abierto. Para problemas lineales de pequeña

o mediana escala, las bibliotecas PuLP y OR-Tools destacan por su versatilidad en la formulación del problema, facilitando la definición de restricciones y variables. Además, permiten el uso de una amplia variedad de optimizadores eficientes, brindando flexibilidad para adaptar la solución según el tamaño del problema. En el caso de problemas de mayor escala y mayor complejidad, el uso de algoritmos de resolución comerciales puede ofrecer ventajas significativas en términos de rendimiento y tiempos de resolución. En particular, para aprovechar al máximo el optimizador CPLEX, se recomienda utilizar directamente la biblioteca IBM CPLEX, ya que ofrece mejor rendimiento y opciones avanzadas de configuración en comparación con su uso a través de bibliotecas de terceros como PuLP o OR-Tools. Los resultados muestran que, dentro de PuLP, los optimizadores libres que ofrecen mejores tiempos de resolución son HiGHS y SCIP, mientras que en OR-Tools, los más eficientes fueron HiGHS y SAT. Por lo tanto, estos dos parecen ser la mejor opción para problemas pequeños o medianos, al combinar facilidad de uso con un buen desempeño en la resolución de problemas lineales y combinatorios.

8. Referencias

Referencias

[1] Infantes Durán, M. (2017). *El problema del viajante (TSP)* [Trabajo de Fin de Grado, Universidad]. Departamento de Estadística e Investigación Operativa.

[2] COIN-OR. (s.f.). *PuLP: Python Linear Programming API*. Recuperado de `https://coin-or.github.io/pulp/`

[3] COIN-OR. (s.f.). *Repositorio de PuLP*. GitHub. Recuperado de `https://github.com/coin-or/pulp/tree/master`

[4] Google. (s.f.). *OR-Tools: Software de optimización de código abierto*. GitHub. Recuperado de `https://github.com/google/or-tools`

[5] Pyomo. (s.f.). *Documentación de Pyomo*. Recuperado de `https://pyomo.readthedocs.io/en/stable/index.html`

[6] SCIP Optimization Suite. (s.f.). *Repositorio de PySCIPOpt*. GitHub. Recuperado de `https://github.com/scipopt/PySCIPOpt`

[7] Diamond, S., & Boyd, S. (s.f.). *CVXPY: Un lenguaje de modelado embebido en Python para optimización convexa*. Recuperado de `https://www.cvxpy.org/`

[8] COIN-OR. (s.f.). *Repositorio del optimizador Cbc*. GitHub. Recuperado de `https://github.com/coin-or/Cbc`

[9] Gurobi Optimization. (s.f.). *Gurobi Optimizer*. Recuperado de `https://www.gurobi.com/`

[10] ERGO-Code. (s.f.). *Repositorio del optimizador HiGHS*. GitHub. Recuperado de `https://github.com/ERGO-Code/HiGHS`

[11] Miranda, M. (2011). *Programación Lineal y su Entorno*. Editorial de la Universidad Católica Argentina.

[12] Mitchell, J. E. (1999). *Branch-and-Cut Algorithms for Combinatorial Optimization Problems*. Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, USA. Revisado el 7 de septiembre de 1999. Recuperado de `http://www.math.rpi.edu/~mitchj`.

[13] Dantzig, G. B., & Wolfe, P. (1959). Decomposition principle for linear programs. *The RAND Corporation*, Santa Monica, California. (Recibido el 24 de noviembre de 1959).

[14] Pérez, A. E. (2016). *Relajación Lagrangiana, métodos heurísticos y metaheurísticos en algunos modelos de Localización e Inventarios*. Máster en Investigación en Matemáticas, Curso 2016-2017.