Desarrollo de herramientas para enseñanza de meta-heurísticas con R.

Baquela, Enrique Gabriel[1], Sebastiani, Iván Guido[2]

[1] *GISOI, Facultad Regional San Nicolás, Universidad Tecnológica Nacional.*

Colón 332 – CP: 2900 – San Nicolás de los Arroyos – Buenos Aires - Argentina.

ebaquela@frsn.utn.edu.ar

[2] *isebastiani@frsn.utn.edu.ar*

Resumen

En el presente trabajo se plantea la utilización de modelos interactivos para la enseñanza de meta-heurísticas en carreras de ingeniería, utilizando para ello una plataforma basada en R.

Se desarrolló una aplicación en R para resolución de problemas de optimización matemática que permite estudiar cada uno de los pasos de la meta-heurística utilizada. La aplicación hace uso de paquetes pre-existentes para optimización en R, re-implementando las funciones adecuadas de manera de poder trazar el proceso. Esto, combinado con una interfaz genérica para definir el espacio de soluciones y la función a optimizar, permite analizar como el algoritmo de optimización selecciona soluciones y se acerca al valor óptimo de la función evaluada.

Dicha aplicación se desarrolló en el marco de la cátedra “Técnicas de optimización de procesos”, iniciándose un estudio sobre la bondad de esta herramienta de enseñanza.

Palabras claves

Optimización; Metaheurísticas; R; Enseñanza.

1. Introducción

Los métodos de optimización son una herramienta fundamental de la Investigación Operativa y de la Ingeniería Industrial. Sin embargo, al ser el diseño de algoritmos un tema no tan afín con la Ingeniería Industrial, la enseñanza de las distintas técnicas y como impactan en las soluciones halladas para cada tipo de problema presenta no pocas dificultades. En este trabajo, proponemos una herramienta que desarrollamos para el dictado de "Técnicas de optimización de procesos", que se complementa con la explicación teórica del funcionamiento del algoritmo y su posterior implementación en problemas prácticos. La herramienta consiste en el paquete "*learnOptim*" [1], una biblioteca para el lenguaje de programación R [2], que permite visualizar la dinámica de un determinado algoritmo de optimización y evaluar la bondad de las soluciones devueltas por el mismo ante escenarios cambiantes.
R es un poderoso lenguaje de programación orientado a la matemática en general y estadística en particular, con miles de paquetes desarrollados por la comunidad de usuarios que amplían sus capacidades. Es free-software, por lo cual no posee costos de licencias ni limitaciones de capacidad, además de poder ser ampliado a voluntad (con los conocimientos necesarios, obviamente). Sus principales ventajas consisten en poseer una sintaxis más cercana al formalismo matemático que a un lenguaje de programación convencional, una gran capacidad para manejar conjuntos de datos en vez de datos individuales y un poderoso motor de graficación.
"*learnOptim*" se basa en paquetes pre-existentes de optimización, invocando o recreando sus funciones, incluyendo en la misma sistemas de monitorización, a fin de poder obtener medidas del desempeño del mismo.

2. Desarrollo

El paquete "*learnOptim*" está basado en el uso de objetos visores que relevan información sobre la evolución del algoritmo para mostrar los resultados al usuario al finalizar la optimización o durante la misma. El alumno puede, amén de la descripción teórica del funcionamiento del algoritmo y la aplicación del mismo en casos prácticos, ver cómo funciona dicho algoritmo, analizar que tan bueno es a la hora de generar soluciones y cuantos pasos realiza hasta hallar la solución óptima, entre otros.
Además de la visualización de los algoritmos, la biblioteca también permite analizar la bondad de la solución óptima hallada ante cambios en los escenarios. Para ello se implementó un módulo basado en Montecarlo para evaluar la función objetivo ante variaciones en los parámetros.

2.1. Arquitectura genérica de una función de optimización en learnOptim

Una función de optimización genérica de "*learnOptim*" presenta la estructura general del algoritmo de optimización, con dos agregados: un "*Monitor*" y un "*Interacturador*". El "*Monitor*" es el encargado de relevar las soluciones parciales halladas por el algoritmo, a fin de poder mostrar la evolución del mismo. El "*interactuador*" presenta una interfaz para seguir paso a paso el cambio de estado del algoritmo, avanzando el mismo mediante comandos del usuario.

Algoritmo 1 – *Función genérica de optimización*

```
funcionOptimizacion(sentido, funcionObjetivo, funcionEvalRestricciones, parametros){
        solActual <- generarSolInicial
        agregarSolucion(monitor, solActual)

        Hasta que (solOptima=Verdadero){
                # Procesos propios de una única iteración del algoritmo de optimización
                agregarSolucion(monitor, solActual)
        }
        Devolver Lista(funcionObjetivo(solActual), solActual, monitor)
}
```

En el pseudo-código anterior podemos ver una simplificación de la estructura de las funciones de optimización de "*learnOptim*". Todas las funciones tienen dos parámetros obligatorios, el sentido (maximizar o minimizar) y la función objetivo a optimizar (definida según el formalismo de modelado de problemas presentado en *2.2*). Opcionalmente, puede incluir una función de evaluación de

restricciones y una lista de parámetros adicionales del algoritmo (por ejemplo, tamaño del paso en un Hill Climbing).

El resultado de llamar a la función de optimización es una lista compuesta por el valor de la función objetivo, los valores de las variables de decisión y el objeto monitor, el cual permite el análisis del desempeño del algoritmo.
Amén a la estructura para generar el monitor, cada función de optimización contiene inserciones dentro de los procesos propios del algoritmo del siguiente estilo:

Algoritmo 2 – *Interacción de función de optimización con interactuador*

```
Paso_01
mostrarSalidaPaso_01
Paso_02
mostrarSalidaPaso_02
…
Paso_n
mostrarSalidaPaso_n
```

Donde las funciones del tipo "*mostrarSalidaPaso_n*" devuelven información relativa a la etapa en la cual se encuentra el algoritmo.

2.2. Modelado de problemas

A fin de presentar una interfaz homogénea, los problemas se modelan mediante funciones de R. Dichas funciones deben tener como primer argumento el vector de las variables independientes y uno o más vectores de parámetros de la misma (los coeficientes de la función objetivo o de las restricciones, según sea el caso). Para cada problema es necesario contar con una función objetivo (la cual devuelve un valor numérico) y opcionalmente una función de evaluación de restricciones (que devuelve verdadero o falso según se cumplan o no las restricciones).

Algoritmo 3 – *Definición de funciones objetivos (arrba) y de control de restricciones (abajo)*

```
funcObjetivo <- function(vectVar, vectParams){
        # Calculo del valor de la función objetivo
}

funcRestricciones <- function(vectVar, vectCoef, vectTermIndep, vectSentido){
        # Análisis de restricciones
}
```

Por ejemplo, para un problema lineal, la definición de las funciones podría ser:

Algoritmo 4 – *Ejemplo de función objetivo lineal (arrba) y de control de restricciones lineales (abajo)*

```
funcObjetivo <- function(vectVar, vectParams){
        sum(vectVar * vectParams)
}

funcRestricciones <- function(vectVar, vectCoef, vectTermIndep, vectSentido){
        ladoIzquierdo <- sum(vectVar * vectCoef)
        chech(ladoIzquierdo, vectSentido, vectTermIndep)
}
```

Dentro del paquete incluimos la función "*check()*" que permite evaluar si se cumplen o no todas las restricciones, por lo cual en la definición de las mismas solo es necesario explicitar la forma de cálculo del lado derecho.

2.3. Análisis de iteraciones

En base al objeto monitor generado, es posible analizar como el algoritmo de optimización selecciona y mejora soluciones. Cada registro de la tabla se corresponde con una iteración, por lo tanto, graficando los valores de la función objetivo podemos observar cómo evoluciona el algoritmo.

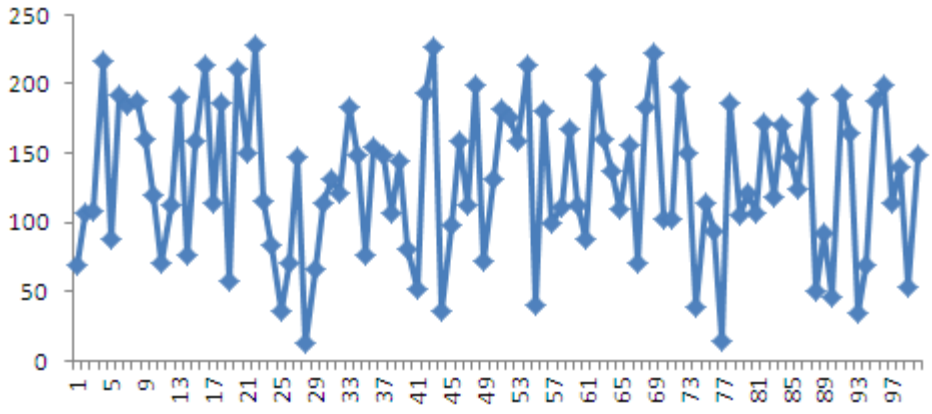

Figura 1 – *Evolución de la función objetivo en cada iteración del algoritmo*

También podemos graficar los valores de la función objetivo encontrados, no en función de las iteraciones, sinó en función de las variables de decisión (para el caso de problemas con una o dos variables de decisión).

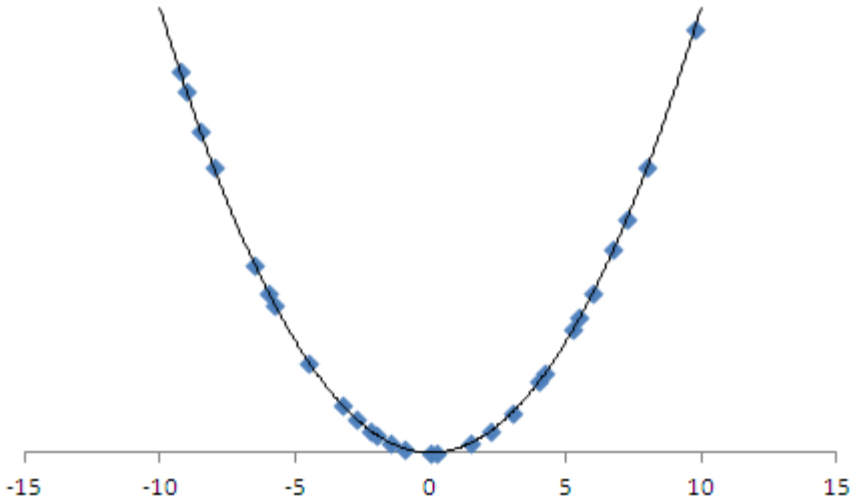

Figura 2 – *Valores de función objetivo probados vs función objetivo*

Y, obviamente, visualizar la selección sobre el conjunto solución (para hasta tres variables de decisión):

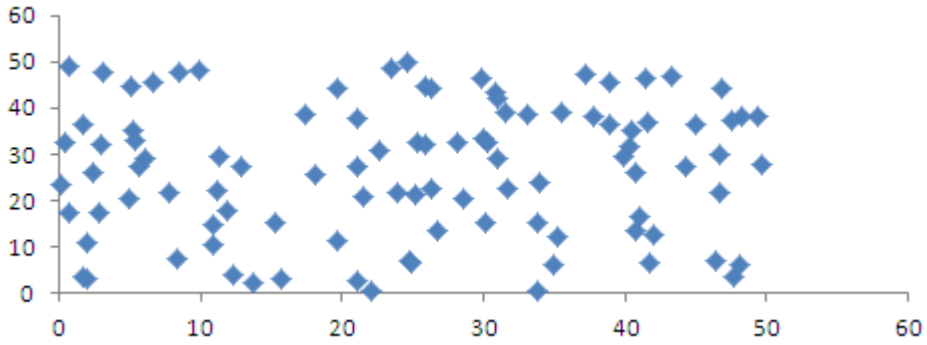

Figura 3 – *Soluciones evaluadas*

2.4. Análisis de factibilidad

Debido a que a nuestras funciones le pasamos como parámetro una función de evaluación de las restricciones, el objeto monitor indica, para cada solución, si la misma fue factible o no. En base a ello, podemos graficar la proporción de los mismos.

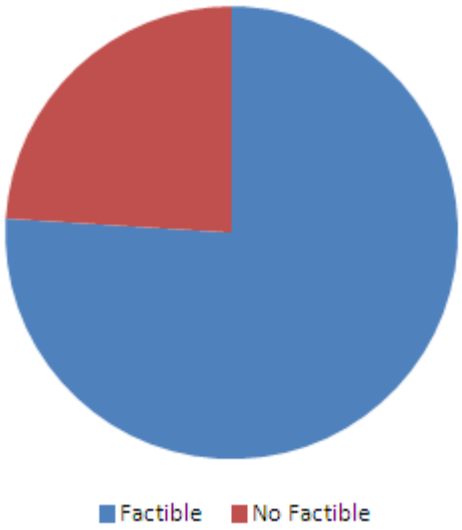

Figura 4 – *Proporción de soluciones factibles y no factibles exploradas*

2.5. Análisis de sensibilidad

"*learnOptim*" incorpora una herramienta basada en Montecarlo para evaluar la sensibilidad de la solución hallada. El algoritmo de análisis es el siguiente:

Algoritmo 4 – *Evaluación de escenarios por Montecarlo*

```
evaluacionEscenario(vectSolución, funcionObjetivo , vectCotasParametros, cantIteraciones){
        evaluaciones <- vector()
        for i=1 to cantIteraciones{
                vectEscenario <- generarUniforme(vectCotasParametros)
                agregarEvaluacion(evaluaciones, funcionObjetivo(vectSolucion, vectEscenario))
        }
        Devolver evaluaciones
}
```

Donde "*evaluaciones"* es un vector con valores de la función objetivo ante diversos escenarios, y "*vectEscenarios*" es un vector que almacena valores de los parámetros de la función objetivo generador aleatoriamente por medio de una distribución uniforme (acotada entre los valores de "*vectCotasParametros*").

3. Implementación en clase

El paquete "*learnOptim*" se introdujo en 2013 en la cátedra "Técnicas de optimización" en la carrera de Ingeniería Industrial de la FRSN-UTN. Su uso fue complementario a la etapa de enseñanza del funcionamiento de los algoritmos, estudiando los alumnos las características, analizando su funcionamiento mediante "*learnOptim*" y por último resolviendo problemas complejos con "*learnOptim*" o con paquetes especializados (y en general, más eficientes, como "*lpSolve*").
La respuesta de los alumnos fue buena. Al finalizar el curso, en la evaluación final por parte de los alumnos acerca de la materia, el aspecto mejor evaluado fue la posibilidad de ver como trabajan los algoritmos utilizados (junto con el enfoque hacia problemas reales de la cátedra).

4. Desarrollos futuros:

El paquete “learnOptim” está en desarrollo. Actualmente soporta los siguientes algoritmos:

- Búsqueda Aleatoria
- Simplex
- Hill Climbing
- Algoritmo Genético Binario

Se prevé, en el corto plazo, desarrollar algoritmos de recocido simulado, búsqueda tabú y optimización por enjambres.

5. Conclusiones

Si bien la implementación de “*learnOptim*” en las cátedras comenzó en 2013, y faltan algunos cursos mas para terminar de depurarlo, creemos que se trata de una herramienta muy útil como complemento a las técnicas tradicionales de enseñanza de la Investigación Operativa en Ingeniería Industrial.

6. Referencias

[1] Baquela, E (2013). “learnOptim”.http://www.modelizandosistemas.com.ar/p/learnOptim.html

[2] R Core Team (2012). “R: A language and environment for statistical computing”. *R Foundation for Statistical Computing, Viena, Austria.* *Http://www.R-project.org*

[3] Reiners, T., Voß, S (2004). *“Teaching meta-heuristics within virtual learning environments”*. International Transactions in Operational Research. 11, 2, 225-238.

[4] Calvo, A., Cortés, A. et al. (2008). *“Using Metaheuristics in a Parallel Computing Course”*. Lecture Notes in Computer Science. 5102, 659-668.