Creación de solucionadores para el bot.

Mauricio Salim Mahmud Sánchez C312

December 12, 2022

Abstract

Tutorial para la creación de Solucionadores a problemas de la asignatura Modelos de Optimización usando el bot de telegram.

Pasos a seguir para crear un solucionador

1. Seleccionar el problema en cuestion, como por ejemplo los de las clases practicas.

2. Teniendo el problema, localizar las siguientes partes escenciales:

   - `name`: Este seria el ID asociado al problema, es importante verificar que sea único respecto a los demas problemas registrados ya que será con lo que se identificará el solucionador. Ej: `got1`, `pizza1`.
   - `title`: Es el titulo del problema en cuestion.
     Ej: `Game of Thrones I`, `Pizzas Tantricas`
   - `description`: Es la descripción (u orientación) del problema en cuestion, es importante dar lujo de detalles, ya que la idea es que el estudiante resuelva el problema a partir de esa descripción.
   - `default_parameters` : son los parametros por defecto del problema, es importante tener a mano un identificador asociado a un valor reconocible del problema, cosas como:
     {`precio_de_tela: 30`, `cantidad_de_horas_sin_luz:18`,...}, respetando la forma llave valor,(como un `dict` de python)
   - `variables` Al igual que los parametros, serán las variables reconocibles del problema, que son las que el usuario final (el estudiante) va a calcular por su cuenta y las comparará con la solución dada

por nuestro solucionador. Es necesario que de cada variable se de una descripción.

3. Creamos un archivo de Python, en este caso donde estan los Solucionadores almacenados (por defecto es en `/solvers`). Este archivo **tiene** que ser el nombre que escogimos mas arriba como identificador del problema seguido por la extensión de python, Ej: `got1.py`

4. En el archivo creado de python vamos a importar la clase base de nuestro solucionador previamente implementada. Ademas importamos la libreria de optimización de python, en este caso GEKKO

```
from BaseSolver import BaseSolver
from gekko import GEKKO
```

5. Procedemos a implementar el solucionador, que hereda de `BaseSolver` y le definimos las caracteristicas básicas descritas en el paso 2.

```
class SolverName(BaseSolver):
    # name es el identificador
    # y nombre del archivo de python
    _name=name

    # title es el titulo asociado al problema
    _title=title

    # description es la descripcion del problema
    _text=description

    # default_parameters es un diccionario que tiene
    # la forma Dict[str,Tuple[Any,str]] siendo la
    # llave el nombre del parametro y el valor una
    # tupla con el valor del parametro por defecto y
    # una descripcion destianda al usuario final
    _default_parameters = default_paramenters

    # variables es un diccionario de la forma
    # Dict[str,Tuple[str,Any | None]] donde la llave
    # es el nombre de la variable y el valor es una
    # tupla donde el primer valor es la descripi-
    # cion de esta y el otro es como se va a mostrar
    # el valor por defecto de la variable ante el
    # usuario final, dejalo en None para que este por
    # defecto.
    _variables_descriptions = variables
```

Luego de esto ya tendríamos los datos del problema listos.

Ahora, falta la implementación del solucionador.

6. El solucionador que estamos haciendo necesita que le sobrescriban dos métodos de la clase base `BaseSolver`

   Estos métodos son:

   - Calcular la mejor solucion:

   ```
   def _solve_model(self) -> Dict[str, Any]:
       # calcula los valores de las variables
       # con los que se maximiza la funcion
       # objetivo.
       pass
   ```

   Es necesario que retorne un `dict` de python donde las llaves son el nombre de las variables y los valores son los que maximizan la función objetivo.

   - Comparar la mejor solución con la del usuario (el estudiante):

   ```
   def _compare_solution(
       self,
       solution: Dict[str, Any],
       best_solution: Dict[str, Any]
   ) -> None:
       # se compara `solution` (la solucion del
       # usuario) con `best_solution` (la solucion
       # calculada en `_solve_model()`)
       pass
   ```

   Este método es el encargado de ver si los valores de las variables dadas por el usuario tienen lógica y son correctos, de ser correctos debe hacer una comparación con la mejor solución, calculada previamente.

7. Entonces procedemos a, usando GEKKO (u otra libreria de optimización de python), modelar la solucion del problema en cuestión. Esto se va a hacer dentro del método `_solve_model()` visto en el paso anterior. Es importante retornar el `dict` relacionado a la mejor solución.

8. La clase base `BaseSolver` tiene métodos ya implementados con el objetivo de dar valoración al usuario final sobre su solución, estos son:

   - `_log_message(self, message)`

     Guarda un mensaje de solución para el usuario final, usualmente relacionado con la comparación de su solución con la optima.

- `_log_error(self, message)`
  Guarda un mensaje de error relacionado con la solución del usuario, suele usarse cuando se viola alguna restricción del problema en cuestión por parte de la solución del usuario. Después de usarse se suele terminar la ejecución de la comparación entre soluciones.

9. Estos métodos vistos anteriormente se usan dentro del método `_compare_solution()` mientras se hacen las comparaciones con la mejor solución para dar una valoración predeterminada al usuario final.

10. Luego ya tenemos implementado el nuevo solucionador, ahora solo hay que agregar al final del archivo la siguiente linea:

```
    defaultSolver =SolverName
```

11. Despues de esto ya debería de estar agregado al bot para su proxima ejecución.

Luego de seguir estos pasos el código del archivo debería de quedar algo así:

```
from BaseSolver import BaseSolver
from gekko import GEKKO

class SolverName(BaseSolver):
    _name=name
    _title=title
    _text=description
    _default_parameters = default_paramenters
    _variables_descriptions = variables

    def _solve_model(self) -> Dict[str, Any]:
        # cuerpo de la funcion
        pass

    def _compare_solution(
        self,
        solution: Dict[str, Any],
        best_solution: Dict[str, Any]
    ) -> None:
        #cuerpo de la funcion
        pass

defaultSolver = SolverName
```

Solucionador de ejemplo.

```
from gekko.gekko import GKVariable
from BaseSolver import BaseSolver, UserSolution
from typing import Dict, Any
from gekko import GEKKO

class GotSolver(BaseSolver):
  _name="got1"
  _title="Game␣Of␣Thrones␣I"
  _text="""
    ***Texto relacionado con el problema,
     lo omitimos aqui porq era largo.
  """
  _default_parameters = {
    "hierro":(600000,"Cantidad␣de␣Hierro"),
    "madera":(400000,"Cantidad␣de␣Madera"),
    "cuero":(800000,"Cantidad␣de␣Cuero"),

    "c_h_sword":(10,"Costo␣de␣hierro␣de␣las␣espadas"),
    "c_m_sword":(2, "Costo␣de␣madera␣de␣las␣espadas"),
    "c_c_sword":(4, "Costo␣de␣cuero␣de␣las␣espadas"),

    "c_h_bow":(2, "Costo␣de␣hierro␣de␣los␣arcos"),
    "c_m_bow":(10,"Costo␣de␣madera␣de␣los␣arcos"),
    "c_c_bow":(5, "Costo␣de␣cuero␣de␣los␣arcos␣"),

    "c_h_catapult":(30, "Costo␣de␣hierro␣de␣las␣catapultas"),
    "c_m_catapult":(100,"Costo␣de␣madera␣de␣las␣catapultas"),
    "c_c_catapult":(50, "Costo␣de␣cuero␣de␣las␣catapultas␣"),

    "sword_damage":(15,"Dano␣de␣las␣espadas"),
    "bow_damage":(10,"Dano␣de␣los␣arcos"),
    "catapult_damage":(8,"Dano␣de␣las␣catapultas")
  }
  _variables_descriptions= {
    "amount_swords":("Cantidad␣de␣espadas.",None),
    "amount_bows":("Cantidad␣de␣arcos.",None),
    "amount_catapults":("Cantidad␣de␣catapultas.",None)
  }

 def _solve_model(self):
   m = GEKKO(remote=False) #Initialize gekko

   m.options.SOLVER = 1

   iron_units= m.Param(value=self.get_param_value("hierro"))
   wood_units= m.Param(value=self.get_param_value("madera"))
```

```
    leather_units= m.Param(value=self.get_param_value("cuero"))

    amount_swords    = m.Var(lb = 0,integer=True)
    amount_bows      = m.Var(lb = 0,integer=True)
    amount_catapults = m.Var(lb = 0,integer=True)

    #costos
    c_h_sword = self.get_param_value("c_h_sword")
    c_m_sword = self.get_param_value("c_m_sword")
    c_c_sword = self.get_param_value("c_c_sword")

    c_h_bow = self.get_param_value("c_h_bow")
    c_m_bow = self.get_param_value("c_m_bow")
    c_c_bow = self.get_param_value("c_c_bow")

    c_h_catapult = self.get_param_value("c_h_catapult")
    c_m_catapult = self.get_param_value("c_m_catapult")
    c_c_catapult = self.get_param_value("c_c_catapult")

    #Iron
    m.Equation(amount_swords*c_h_sword+amount_bows\
     *c_h_bow+amount_catapults*c_h_catapult<=iron_units)
    #Wood
    m.Equation(amount_swords*c_m_sword+amount_bows\
     *c_m_bow+amount_catapults*c_m_catapult<=wood_units)
    #Leather
    m.Equation(amount_swords*c_c_sword+amount_bows\
     *c_c_bow+amount_catapults*c_c_catapult<=leather_units)

    #funcion objetivo
    m.Maximize(self.total_damage(
     amount_swords,amount_bows,amount_catapults
    ))

    m.solve(disp = False)

    return {
      "amount_swords": amount_swords.VALUE[0],
      "amount_bows": amount_bows.VALUE[0],
      "amount_catapults": amount_catapults.VALUE[0],
      #en ese caso se retorno ademas el resultado
      #de la funcion objetivo maximizado, no es necesario.
      "obj": self.total_damage(
         amount_swords.VALUE[0],
         amount_bows.VALUE[0],
         amount_catapults.VALUE[0])
    }

  def _compare_solution(
```

```
    self,
    solution: Dict[str,Any],
    best_solution: Dict[str,Any]):

    #totales
    t_hierro=self.get_param_value("hierro")
    t_madera=self.get_param_value("madera")
    t_cuero=self.get_param_value("cuero")

    #costos
    c_h_sword = self.get_param_value("c_h_sword")
    c_m_sword = self.get_param_value("c_m_sword")
    c_c_sword = self.get_param_value("c_c_sword")

    c_h_bow = self.get_param_value("c_h_bow")
    c_m_bow = self.get_param_value("c_m_bow")
    c_c_bow = self.get_param_value("c_c_bow")

    c_h_catapult = self.get_param_value("c_h_catapult")
    c_m_catapult = self.get_param_value("c_m_catapult")
    c_c_catapult = self.get_param_value("c_c_catapult")

   #mensajes de error personalizados
   #por si no alcanzan los materiales :(
   sword = solution["amount_swords"]
   t_hierro -= sword*c_h_sword
   t_madera -= sword*c_m_sword
   t_cuero -= sword*c_c_sword
   if t_hierro < 0 or t_madera < 0 or t_cuero < 0:
     self._log_message("No se pueden construir\
tantas espadas :(")
     self._log_message(f"Nos quedamos sin material\
necesitado para las espadas.")
     return

   bow = solution["amount_bows"]
   t_hierro -= bow*c_h_bow
   t_madera -= bow*c_m_bow
   t_cuero -=  bow*c_c_bow
   if t_hierro < 0 or t_madera < 0 or t_cuero < 0:
     self._log_error("No se pueden construir tantos arcos :(")
     self._log_message(f"Nos quedamos sin material necesitado\
para los arcos.")
     return

   catapult = solution["amount_catapults"]
   t_hierro -= catapult*c_h_catapult
   t_madera -= catapult*c_m_catapult
   t_cuero -=  catapult*c_c_catapult
```

```
    if t_hierro < 0 or t_madera < 0 or t_cuero < 0:
      self._log_error("No se pueden construir tantas\
catapultas :(")
      self._log_message(f"Nos quedamos sin material \
necesitado para las catapultas.")
      return

    if(sword < 0 or bow < 0 or catapult < 0):
      self._log_error('Has creado una cantidad negativa,\
el mundo implosiona, mision cumplida,\
los caminantes blancos han muerto')
      return

    #calculo de danno
    damage_dealt =self.total_damage(sword,bow,catapult)
    best_posible =best_solution["obj"]
    self._log_message(f'Hemos realizado {damage_dealt}\
dannos en las tropas enemigas')
    if damage_dealt < best_posible*0.98:
      self._log_message(f'Necesitamos\
{best_posible*0.98-damage_dealt} de danno\
para alcanzar la victoria')
    if(damage_dealt > best_posible*0.98):
      self._log_message('Hemos aniquilado a\
las tropas enemigas,una rotunda victoria')
    elif(damage_dealt > best_posible*0.8):
      self._log_message('Estuvimos tan cerca de\
la victoria, fue un honor luchar a su lado')
    elif(damage_dealt > best_posible*0.5):
      self._log_message('Al menos eliminamos\
la mitad de sus tropas')
    elif(damage_dealt < best_posible*0.1):
      self._log_message('No se como alguien logro\
hacerlo tan mal')
    else:
      self._log_message('Que desastre!')


 def total_damage(self,
  swords: int | GKVariable,
  bows: int | GKVariable,
  catapults: int | GKVariable ):

    s_d: int = self.get_param_value("sword_damage")
    b_d: int = self.get_param_value("bow_damage")
    c_d: int = self.get_param_value("catapult_damage")
    return s_d*swords+b_d*bows+c_d*catapults

defaultSolver = GotSolver
```