Minimod: A Mixed Integer Solver for Spatio-Temporal Optimal Nutrition Intervention

Aleksandr Michuda

`minimod` is a solver written in python that solves the optimal nutrition intervention over space and time. It uses `mip` a mixed integer solver that uses the `CBC` solver, a fast, extensible and open-source linear solver. `minimod` comes with some constraints baked in and the underlying `mip` model is exposed and can be used for adding custom constraints if needed.

The problem that `minimod` solves initially can be written like so:

$$\begin{aligned}
&\text{Max} \sum_k \sum_t Y_{k,t} \sum_j \frac{EfCvg_{kj,t}}{(1+r)^t} \\
&+ \sum_k \sum_j \sum_t X_{k,j,t} \frac{EfCvg_{k,j,t}}{(1+r)^t} \\
&s.t. \\
&\sum_k \sum_t Y_{k,t} \sum_j \frac{\text{TC}_{k,j,t}}{(1+i)^t} \\
&+ \sum_k \sum_j \sum_t X_{k,j,t} \frac{\text{TC}_{k,j,t}}{(1+i)^t} \leq \text{TF}
\end{aligned}$$

which essentially maximizes discounted coverage given a budget constraint. This problem can give solutions that are both time and space specific. The dual problem minimizes discounted costs given a minimum coverage constraint.

Quickstart

In order to run the model, we need to first import `minimod` and `pandas` in order to later load in our data:

```
1 import minimod as mm
2 import pandas as pd
3 import geopandas as gpd
```

```
Using Python-MIP package version 1.6.8
```

Now we load in data and instantiate the model we want (`BenefitSolver` or `CostSolver`).

The solvers take several arguments:

- data -> a pandas dataframe of benefits and cost data. This data needs to be of a certain form, mainly a long form of data.
  - Default: None

k	j	t	benefits	costs
maize	north	0	100	10
maize	south	0	50	20
maize	east	0	30	30
maize	west	0	20	40

- `intervention_col` -> the name of the intervention variable
  - Default: 'intervention'
- `space_col` -> the name of the spatial variable

  - Default: 'space'

- `time_col` -> the name of the time variable
  - Default: 'time'
- `benefit_col` -> the name of the benefit variable
  - Default: 'benefit'
- `cost_col` -> the name of the cost variable
  - Default: 'costs'
- `interest_rate_cost` -> the interest rate on costs
  - Default: 0.0 -`interest_rate_benefit` -> the interest rate on benefits
  - Default: 0.03
- `va_weight` -> The weight to give the benefits during intervention
  - Default: 1.0

For `BenefitSolver`, we also have:

- total_funds -> Maximum Budget
  - Default: 35821703

And for `CostSolver`:

- minimum_benefit -> The Minimum benefit constraint
  - Default: 15958220

Now we load the data and calculate the minimum coverage constraint based on the variable `vasoilold`

```
32 df = pd.read_csv('../../examples/data/processed/example1.csv')
```

```
33
```

```
34 c = mm.CostSolver(minimum_benefit = vasoilold_constraint)
```

```
                MiniMod Nutrition Intervention Tool

                Optimization Method: MIN

                Version: 0.0.3dev

                Solver: CBC
```

We then fit the model:

```
36 c.fit(data = df)
```

```
[Note]: Processing Data...

[Note]: Creating Base Model with constraints

[Note]: Optimizing...

[Note]: Optimal Solution Found
```

The optimal interventions are stored in an attribute available after fitting:

```
38 opt_df = c.opt_df
```

Then we can generate a report

```
40 c.report()
```

```
+----------------------------+----------------------------+

| MiniMod Solver Results     |                            |
```

Method:	MIN
Solver:	CBC
Optimization Status:	OptimizationStatus.OPTIMAL
Number of Solutions Found:	1

No. of Variables:	450
No. of Integer Variables:	450
No. of Constraints	31
No. of Non-zeros in Constr.	882

Minimum Benefit	1.59582e+07
Total Cost	1.87958e+07
Total Coverage	1.8826e+07

Total Costs and Coverage by Year

time	opt_vals	opt_benefit	opt_costs
1	3	1.68539e+06	1.78616e+06
2	3	1.77221e+06	1.87296e+06

```
|      3 |          3 |   1.82944e+06 | 1.96123e+06 |
|      4 |          3 |   1.87922e+06 | 1.98977e+06 |
|      5 |          3 |   1.8401e+06  | 1.95842e+06 |
|      6 |          3 |   1.88294e+06 | 1.93123e+06 |
|      7 |          3 |   1.91036e+06 | 1.95754e+06 |
|      8 |          3 |   1.93643e+06 | 1.98395e+06 |
|      9 |          3 |   1.96151e+06 | 2.00954e+06 |
|     10 |          3 |   1.47362e+06 | 1.37522e+06 |
+------------------+----------+
| Cost per Coverage | 0.998394 |
+------------------+----------+
```

More Complicated Constraints

Oftentimes, it becomes necessary to add additional constraints to the model. This can include assumptions about costs, such that the cost structure of an intervention assumes start-up costs that forces an intervention to persist through time if those start-up costs are paid at some point. Or there may be assumptions about how certain interventions must be rolled out to all or part of a country (such as a national intervention). `minimod` allows for such constraints through the fit method with the `all_space` and `all_time` arguments. If interventions are dependent on start-up costs from certain periods of time, or that an intervention can be rolled out subnationally but to certain larger groups of regions, we can use `time_subset` and `space_subset`, respectively, to specify that.

Let's say that you know that the interventions "cube", "oil" and "maize" must be rolled out nationally and that "maize" and "cube" have startup costs in the first three time periods, so that if those costs are paid in those time periods, then the interventions must take place for the rest of time.

Then we can write the fitter method like so:

```
opt = c.fit(data = df,
            all_space = ['cube', 'oil', 'maize'],
            all_time = ['maize', 'cube'],
            time_subset = [1,2,3]
            )
```

```
[Note]: Processing Data...
[Note]: Creating Base Model with constraints
[Note]: Optimizing...
[Note]: Optimal Solution Found
```

And then we can create another report:

```
c.report()
```

```
+---------------------------+---------------------------+
| MiniMod Solver Results    |                           |
| Method:                   | MIN                       |
| Solver:                   | CBC                       |
| Optimization Status:      | OptimizationStatus.OPTIMAL |
```

| Number of Solutions Found: | 1 |

+---------------------------+---------------------------+

+-----------------------------+------+

| No. of Variables: | 900 |

| No. of Integer Variables: | 900 |

| No. of Constraints | 2798 |

| No. of Non-zeros in Constr. | 7236 |

+-----------------------------+------+

+-----------------+-------------+

| Minimum Benefit | 1.59582e+07 |

| Total Cost | 3.85073e+07 |

| Total Coverage | 3.94465e+07 |

+-----------------+-------------+

+-----------------------------------+--+

| Total Costs and Coverage by Year | |

+-----------------------------------+--+

| time | opt_vals | opt_benefit | opt_costs |

|-------:|-----------:|--------------:|------------:|

| 1 | 3 | 1.68539e+06 | 1.78616e+06 |

| 2 | 3 | 1.77221e+06 | 1.87296e+06 |

| 3 | 3 | 1.82944e+06 | 1.96123e+06 |

| 4 | 3 | 1.86221e+06 | 1.99482e+06 |

| 5 | 3 | 1.89345e+06 | 2.22715e+06 |

```
|      6 |          3 |   1.92312e+06 | 2.05819e+06 |
|      7 |          3 |   1.95113e+06 | 2.08786e+06 |
|      8 |          3 |   1.97774e+06 | 2.1164e+06  |
|      9 |          3 |   2.00336e+06 | 2.14414e+06 |
|     10 |          3 |   2.02859e+06 | 2.3716e+06  |
+--------------------+----------+
| Cost per Coverage | 0.976189 |
+--------------------+----------+
```

Plotting with `minimod`

There are three basic ways `minimod` can plot results: time trends, histograms, and maps.

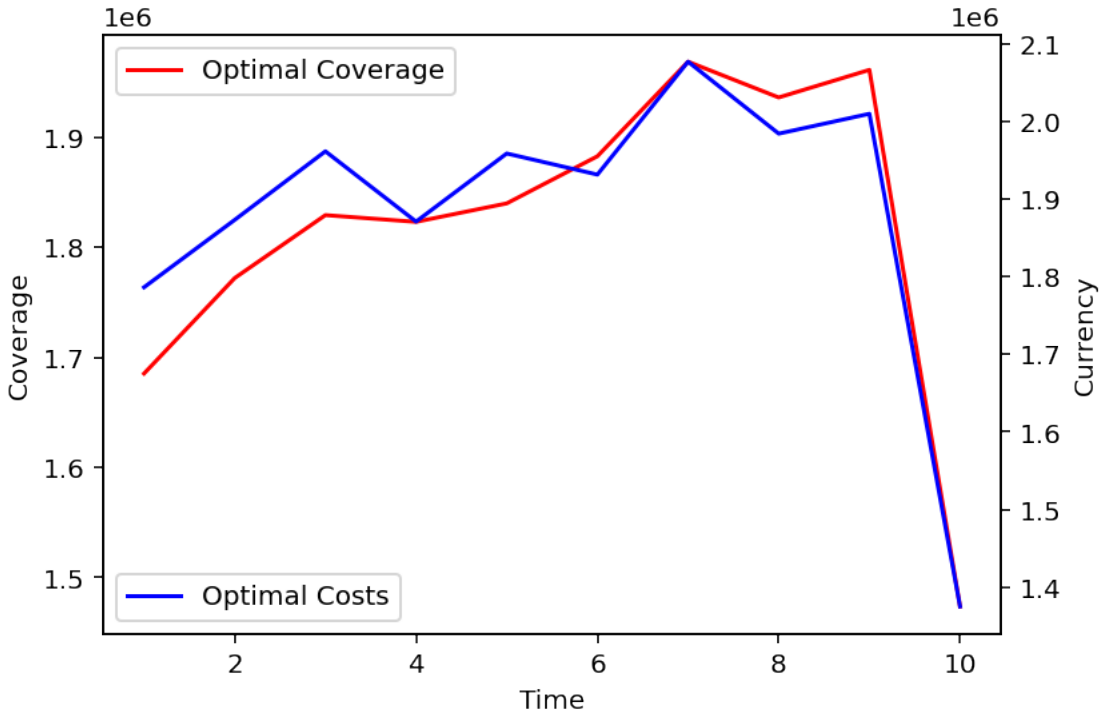
Once optimization has occurred, we can plot the optimal coverage, and optimal costs over time with the `plot_time` method and save it to a directory of our choice:

```
48 c.fit(data = df)
49 c.plot_time(save = "time.png")
```

```
[Note]: Processing Data...
[Note]: Creating Base Model with constraints
[Note]: Optimizing...
[Note]: Optimal Solution Found

(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f627e08e850>,
```

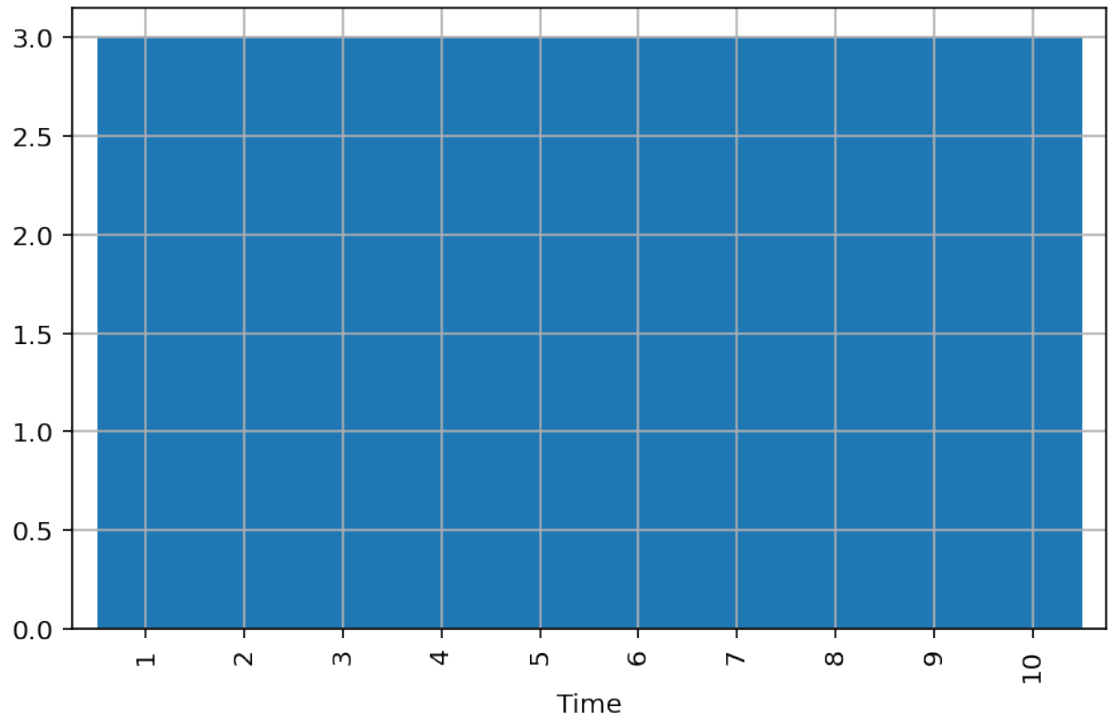
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62bc3b6be0>)
```



Or we can plot the histogram of optimal interventions:

```
50 c.plot_opt_val_hist(save = "hist.png")
```

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f627b335b20>)
```



Plotting Maps

With maps, there are several things that need to be done first: a shape file must be downloaded externally, and then a column in the resulting dataframe (using `geopandas`) must have the same values as the spatial data in the coverage and cost data. In the case of Cameroon, we downloaded a shape file and created the correct column and called it "space":

```
52 # Load data
53 geo_df = gpd.read_file("../../examples/data/maps/cameroon/CAM.shp")
54
55 # Now we create the boundaries for North, South and Cities
56 # Based on "Measuring Costs of Vitamin A..., Table 2"
57 north = r"Adamaoua|Nord|Extreme-Nord"
```

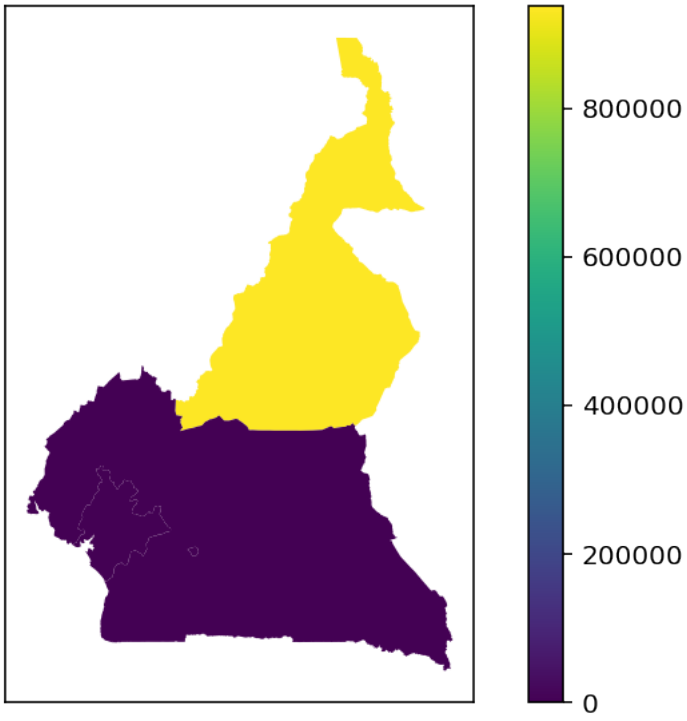
```
south = r"Centre|Est|Nord-Ouest|Ouest|Sud|Sud-Ouest"
cities= r"Littoral" # Duala
# Yaounde is in Mfoundi
geo_df.loc[lambda df: df['ADM1'].str.contains(north), 'space'] = 'North'
geo_df.loc[lambda df: df['ADM1'].str.contains(south), 'space'] = 'South'
geo_df.loc[lambda df: df['ADM1'].str.contains(cities), 'space'] = 'Cities'
geo_df.loc[lambda df: df['ADM2'].str.contains(r"Mfoundi"), 'space'] = 'Cities'

# Now we aggregate the data to the `space` variable
agg_geo_df = geo_df.dissolve(by = 'space')
```

Then we use that data as an input into our map method, `plot_chloropleth`:

```
c.plot_chloropleth(intervention='vasoil',
                   time = [5],
                   optimum_interest='c',
                   map_df = agg_geo_df,
                   merge_key= 'space',
                   save = "map.png")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f627d1985b0>
```



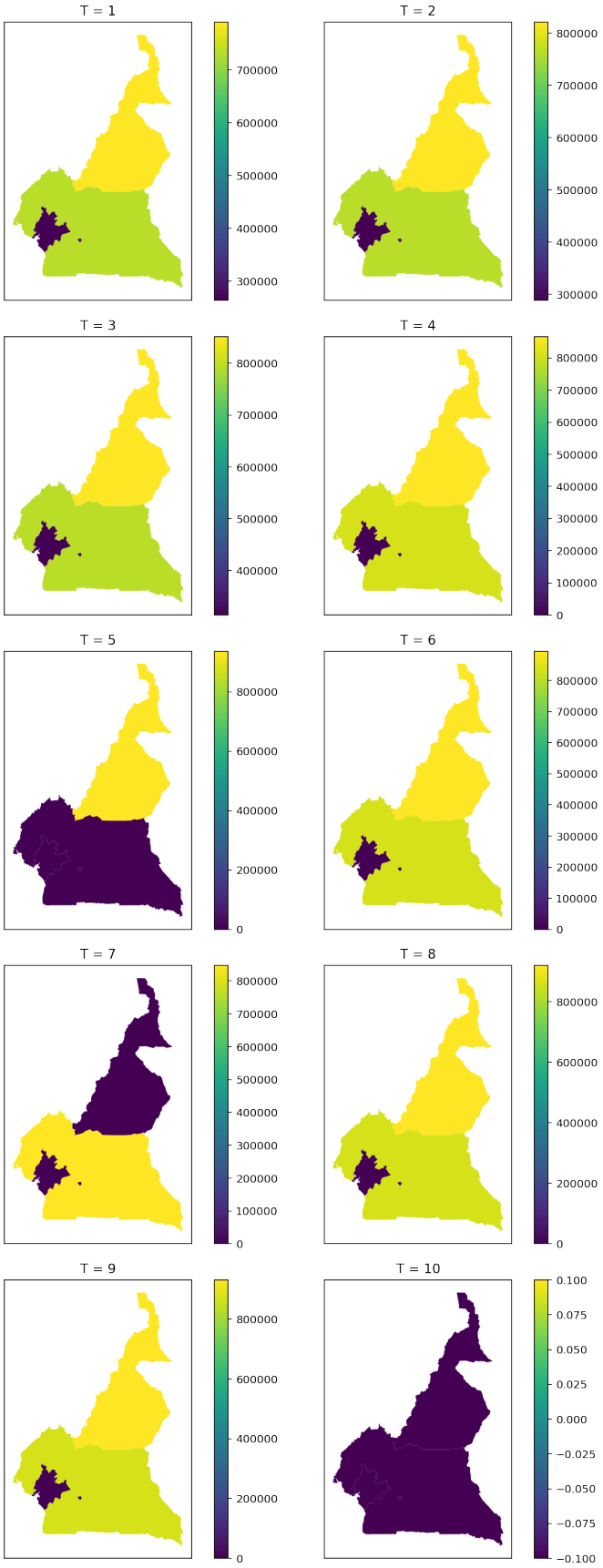
`plot_chloropleth` allows us to choose a particular intervention to map (in this case "vasoil"), at a particular time period, and looking at optimal costs (`optimum_interest = 'c'`), and then save it.

If we want to see how the chloropleth changes through time, we can plot some subset of time, or if we omit the `time` parameter altogether, it will create maps for all time periods.

```
75 c.plot_chloropleth(intervention = 'vasoil',
76                     optimum_interest='c',
77                     map_df = agg_geo_df,
78                     merge_key= 'space',
79                     save = "map2.png")
```

```
(<Figure size 720x1440 with 20 Axes>,
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f627d39d5e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f627d0d59d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f627d07fdf0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f627d0b6280>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f627d0646a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f627d011b80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f627d4e5d90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f627d461730>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f627d1af6a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f62828c6550>]],
      dtype=object))
```


T = 1
T = 2
T = 3
T = 4
T = 5
T = 6
T = 7
T = 8
T = 9
T = 10

A Note about Constraint Choice

The `all_*` constraints described above differ in the python implementation and the GAMS implementation, because I found an inconsistency in the GAMS results that wasn't in the spirit of what I though the constraint was for. To illustrate, here is a simplified example:

In GAMS the relevant constraints are coded like so:

```
yescubeeq(j,t)..        yescube(j,t) =e= sum((cubek),x(cubek,j,t)) ;


allcubeeq(j,jj,t)..           yescube(j,t) =e= yescube(jj,t) ;
```

where j and jj are aliases for space.

So in terms of math, this is what I think the all cube constraint is saying, and let's assume that there are just two cube interventions, $c1$ and $c2$:

$$x_{c1,j,t} + x_{c2,j,t} = x_{c1,jj,t} + x_{c2,jj,t} \forall j \neq jj$$

In this case, since x is binary, there are three possible options:

Possibilities	Sum
Both 1	2
$x_{c1,j,t} = 1$ and $x_{c2,j,t} = 0$	1
$x_{c1,j,t} = 0$ and $x_{c2,j,t} = 1$	1
Both 0	0

If my understanding of the constraint is correct, its function is to say that "if a cube (so one of cube, vascube, oilcube, etc. . .) interventions is used in some region, then it must be used in all regions." In that case, the constraint should be written:

$$x_{c,j,t} = x_{c,jj,t} \text{for } c \in \{c1, c2\} \forall j \neq jj$$

If that's the case, when the variables are both 0 or both 1, that constraint works, but when one of them are 1 and the other 0, it you could have a situation when $x_{c1,j,t} = 0$, but $x_{c1,jj,t} = 1$, for example. This is a way that optimization can satisfy the constraint, but not actually satisfy what the constraint should be doing.

I have decided to implement it in the way described at the beginning:

$$x_{c1,j,t} + x_{c2,j,t} = x_{c1,jj,t} + x_{c2,jj,t} \forall j \neq jj$$