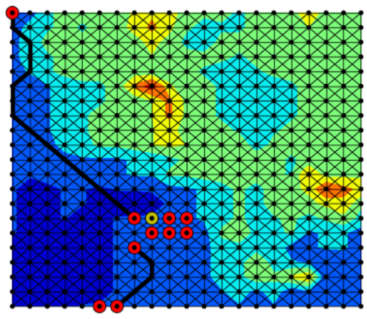
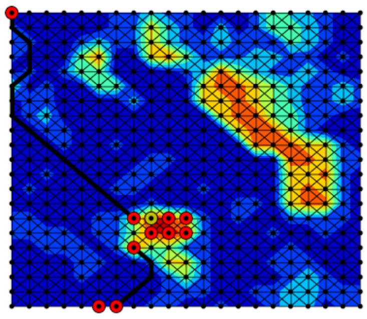
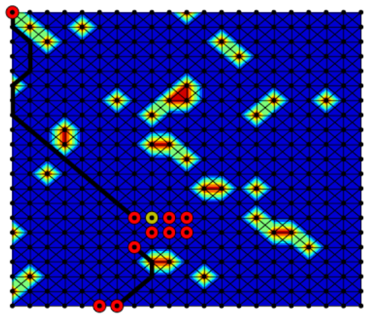
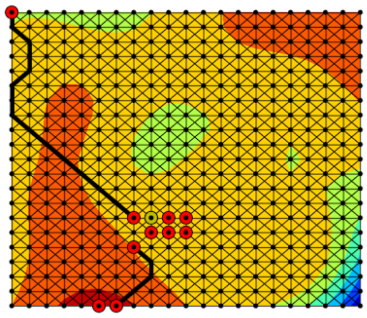
Optimal Energy Grid Prediction

Producing an energy profile for Arizona's geography and energy need









Thurston Sexton
Cole North
Lon Saline

MAE 598 Design Optimization
May 5, 2015

Abstract

America's power grid is old and, while being constantly improved, could be altered to use the optimal energy production in key regions. For example, obtaining energy from solar during the day and coal at night, wind energy in mountainous regions and nuclear energy in flatlands. Creating a system that uses the right energy at the right time would both be cost effective and responsibly green. This project attempts to take a small section of populated land and optimize locations and types of energy production to best fit the needs of the residents, cost restraints, and the environment. The optimization resulted in the creation of one 9.36 TWh coal power plant, eight 720 GWh wind farms, and two 1.08 TWh solar arrays constructed to meet the population's needs.

Table of Contents

Introduction

Motivation

The energy needs of a population grow with the increase in population and the technological advancements of the culture. The ability to determine the best ways to spend finite resources on the production of energy would maximize the effectiveness and efficiency of planning and implementing creation of production centers. Knowledge of the best location of wind turbine farms, nuclear plants, coal fire plants, and solar arrays all in concert would be invaluable to a economically prudent population.

Problem Description and Setup

The system design project involves the creation of an iterative model for a given inhabited geospatial region divided into a rectilinear mesh. Each element of the mesh would include terrain and environmental constants in addition to environmental variables (e.g. wind, solar data, etc.). The model would take the regional elements and optimize an energy grid. The grid would be made up of differing forms of energy production both finite and renewable depending on the regional constants and variables stated above along with other variables of important measures (cost, public attitude, etc.)
A model was developed to determine the best type, number, and location of energy production plants, be it nuclear, solar, wind, or coal. The optimization will be on an iterative system which will determine the best combinations for three different subsystems, and will attempt to improve the long-term viability of the energy system provided by the initial economic constraints of the given populations energy requirements. Building sustainable plant types will provide long-term happiness and low costs, but initially will be expensive and may have less output or poorly timed output compared to coal/gas. Farther out plants will be less of an eyesore and may take advantage of good terrain for building, but will cost more to transport energy from.

Figure 1, below, shows the geographic constraints put on the system. A monetary amount was taken to represent the energy cost needs of the population. This finite allowance was needed to cap the amount of plants the system would be able to build. Within those constraints, locations were chosen as viable or unviable due to proximity costs, environmental factors (wind, water, etc.), the population's attitude towards certain plants, and locations already taken. The networking subsystem then introduced additional constraints on the placement of the plants based on the cost of connecting all the plants into a grid. The final subsystem takes the data from the other two and outputs the best combination of plant types. As stated this was an iterative model and this process runs the iterations until convergence.

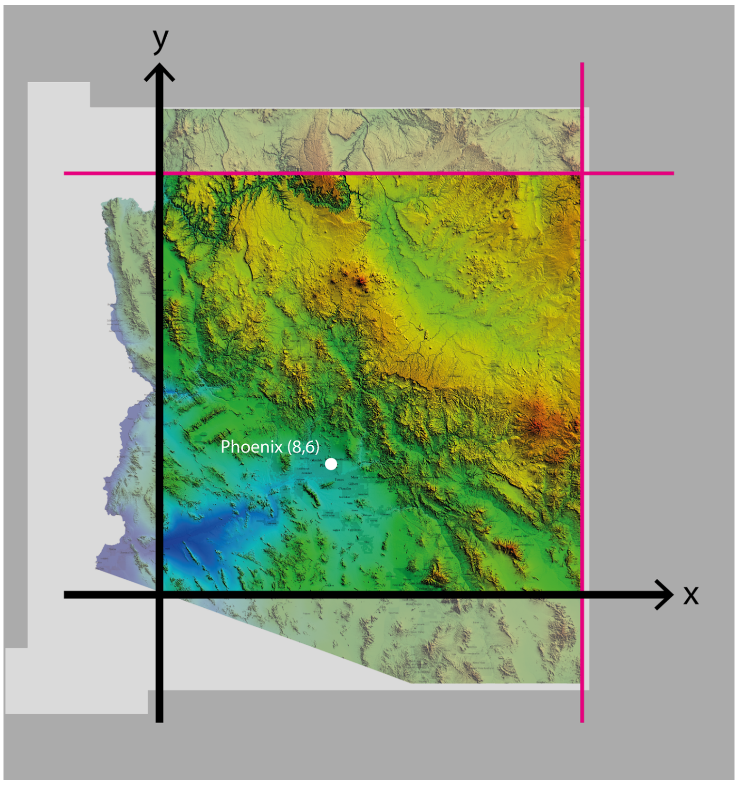
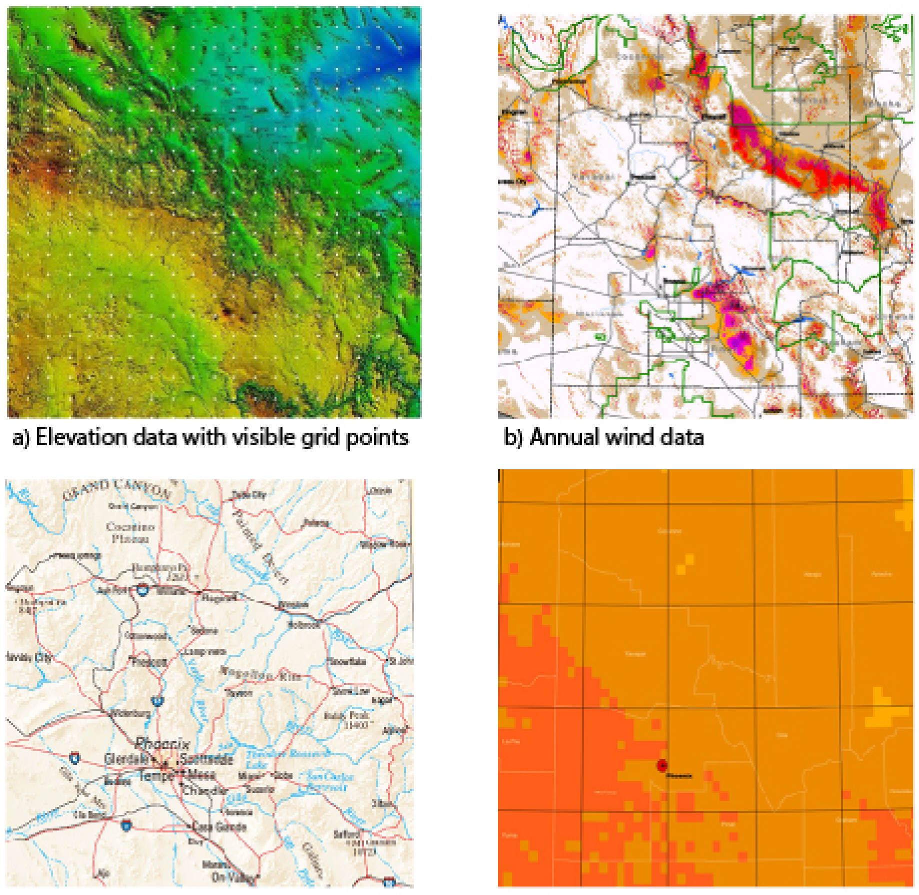


Figure 1. 300 square mile boundary used for the problem.

Plant Locations

The objective of this subsection was to optimize the best possible location for a given number and type of energy production plant. The subsystem would then return the locations to the transmission routing subsystem to be connected in an optimum configuration.

Plant location problems are a fairly common type of problem in design optimization. Usually these problems deal with production of a good and the distribution of said good to various markets. While traditional plant location problems focus on decreasing the distance and cost to produce the goods within some constraint of cost or distance between production centers and markets the present problem looked at other factors associated more closely to each individual plant type. The quality of wind in the geographic area would help determine the best placement for a wind farm. The proximity to water would locate coal and nuclear plants. Yearly sunshine percentages would help guide the placement of solar arrays. The constraints placed on this subsystem were self-imposed boundaries of the largest square (300 square miles) that could be placed within the State of Arizona as shown above in figure 1. These constraints were created to simplify the overall boundary problem. Data was collected at 15 mile intervals within the 300 square miles to create a usable mesh shown below in Figure 2a. Phoenix was taken at the central point in the scheme. The other data sets are also shown graphically in the figure.



a) Elevation data with visible grid points
b) Annual wind data
c) Water location data
b) Annual Sunshine percentage data

Figure 2 a,b,c,d. Graphic representations of data used to optimize plant location..

Due to the immense size of the data sets involved it became necessary to take values only on the grid points then interpolate from that abridged data set back to a continuous data set. The interpolation utilized was a cubic spline interpolation. All data was normalized to ensure that the various data sets could be used in conjunction as weighted decision variables.

Each plant type requires a decision variable(s), with each type, apart from solar, having some form of a penalty variable for proximity to a centralized location where the energy need was,Phoenix. This goodness factor was used to penalize coal and nuclear plants higher the closer these plants were to the central location. Equation 1 shows the logistic function (or sigmoid curve) was used to create the goodness factor.

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}} \qquad \text{Eqn.1}$$

Where L is the curve's maximum value, k is the steepness of the curve, and x_0 is the x-value of the sigmoid's midpoint as shown in Figure 3, below. Table 1 shows the plant types with their respective goodness factor equations.

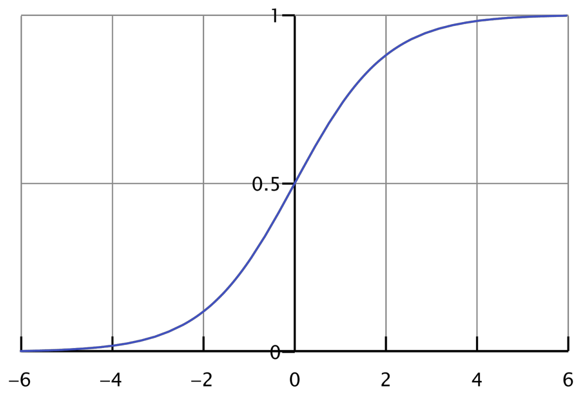


Figure 3. Logistic function used to create the continuous goodness penalty variable.

Table 1. Goodness Factor Equations for given plant types.

Plant Type	Goodness Factor Equation, Gdns
Wind	0
Nuclear	$\left(\frac{4}{15}\right) * [\left(\sqrt{(8-x)^2+(6-y)^2} * 15\right) - 45]$
Coal	$\left(\frac{6}{277}\right) * [\left(\sqrt{(8-x)^2+(6-y)^2} * 15\right) - 138.5]$
Solar	0

Table 2. The objective function(Total), with sub-objective functions for the plant types.

Objective functions	
Wind	$\boldsymbol{f(x,x_w) = \sum_{i=1}^{n}(a * Gdns) + \sum_{i=1}^{n}(b * Dist) + \sum_{i=1}^{n}(c * Wind) + \sum_{i=1}^{n}(Taken\ Locations)}$
Nuclear	$f(x,x_N) = \sum_{i=1}^{n}(a * Gdns) + \sum_{i=1}^{n}(b * Dist) + \sum_{i=1}^{n}(d * Water) - \sum_{i=1}^{n}(Taken\ Locations)$
Coal	$f(x,x_C) = \sum_{i=1}^{n}(a * Gdns) + \sum_{i=1}^{n}(b * Dist) + \sum_{i=1}^{n}(c * Wind) + \sum_{i=1}^{n}(d * Water) - \sum_{i=1}^{n}(Taken\ Locations)$
Solar	$f(x,x_S) = \sum_{i=1}^{n}(a * Gdns) + \sum_{i=1}^{n}(b * Dist) + \sum_{i=1}^{n}(k * Solar) - \sum_{i=1}^{n}(Taken\ Locations)$
Total	$f(x,x_i) = \sum_{i=1}^{n}(Wind) + \sum_{i=1}^{n}(Nuclear) + \sum_{i=1}^{n}(Coal) + \sum_{i=1}^{n}(Solar)$
With x_i=[#Wind, #Nuke, #Coal, #Solar]	

Table 2, above, shows the objective functions used in this subsystem. Due to the systems stochastic nature, genetic algorithms were implemented due to the ability to converge where other methods could not. In the objective functions the constants a,b,c,d were taken as a weight between zero and one to normalize the variable. The Dist variable is a cost penalty given to the location's distance from the central point. Solar, Water, and Wind are the weighted data

parameters addressed earlier. In addition, the taken locations parameter keeps track of previous location placements.
The use of a severe penalty placed on taken locations enabled the algorithm to avoid placing two plants on the same location, in fact the penalty was created in such a way that it helped space the plant locations with respect to all other taken locations and still find an optimum placement.
To reiterate the objective functions were subject to the constraints of the 300 square mile mesh described above.
Priority in the algorithm was given to wind and water data as driving parameters. The method first locates the best wind locations based on geographical wind speed data along with the other parameters. Nuclear power plants are then placed based on the proximity to the central point,Phoenix, and to available water. It should be noted that the water data was only sources of standing water not subterranean water sources e.g. water tables and aquifers. Coal fire plants were then places based on the same distance and water proximity parameters. Finally, solar arrays were located. Solar arrays were located last due to the abundance of possible solar locations in Arizona and the method needed to not fill important possible wind, coal, or nuclear locations with solar plants.

Energy Transmission Routes

A major cost in building any new plant is the cost for transmission lines to reach it. These can be very expensive, up to costing over $1M/mile. To minimize the investment required for plant placement, and to propagate those costs back to the plant choosing algorithm (production strategy), a two-pronged approach was used.

The first part is to determine the minimum cost required for a path between any two primary nodes (a primary node being either a power plant or Phoenix itself). Though distance is the primary concern in cabling placement, other factors like obstruction from large bodies of water or sharp changes in elevation can also increase costs. The previously described topological grid was used to create a graph in the Python module NetworkX, with nodes located at the centers of grid spaces. Then, a 'king's graph' was constructed, such that every node was connected to 8 neighbors (up, down, left, right, and diagonal grid spaces).

To use a shortest-path minimization on a graph, the weights of the edges between neighbors is needed. Three weight types, added linearly for each edge, were used here:

1. Distance (15.0mi for rectilinear edge, 21.1mi for diagonals)
2. Large bodies of water (5mi-equivalent penalty for two consecutive water nodes)
3. Incline (Distance*[absolute normalized gradient between nodes])

The incline penalty assumed that the steepest incline in AZ would double the transmission cost per mile, and scaled all inclines accordingly. An approximate heat-map for the total penalty for the considered region is shown in figure 3. From this network, the shortest-path problem can be efficiently solved for any pair of nodes by any greedy Algorithm like Dijkstra's algorithm or A*.

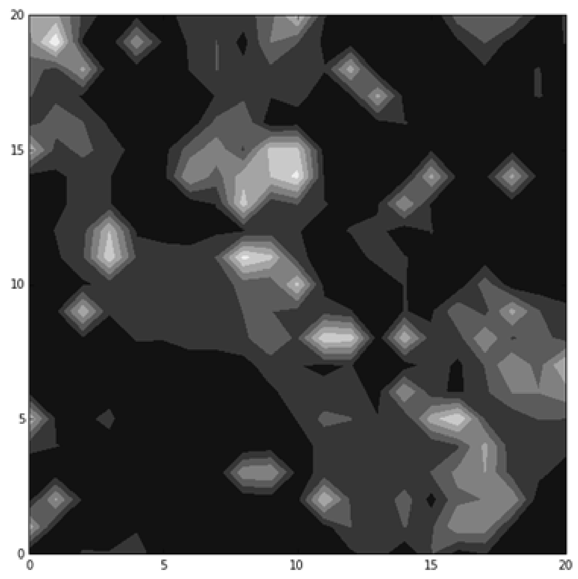


Figure 3. Approximate pathing penalty map for AZ, where white is high penalty.

The primary goal here is to minimize the total transmission cost, however, so the decision of which pairs are required in order to reach all nodes on a single network is equally important. This is commonly referred to as the Minimum Spanning Tree problem. Kruskal's Algorithm can solve the MST problem for a set of nodes, returning a minimum-length tree ensuring all nodes are connected. To do this efficiently, a helper graph was created, with only Primary nodes, where the weights between each pair was determined using the A* algorithm described above. The complete graph, with regular degree n-1 and n(n-1)/2 edges, was therefore weighted to approximate real topology of AZ. Then, Kruskal's algorithm was applied on this helper graph, returning a Tree that only had the minimum number/weights of edges. Therefore, the total cost to route cable between all plants and phoenix would have the combined weight of all edges in this MST.

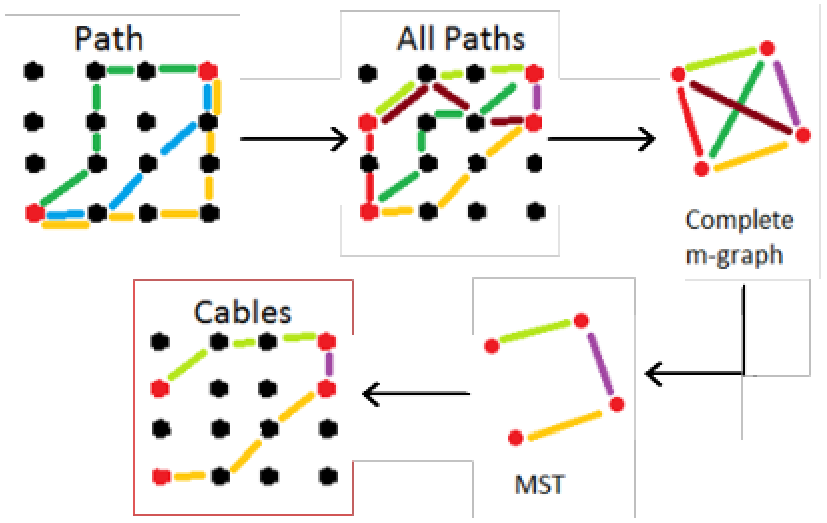


Figure 4. A summary of the process to determine minimum cable cost.
1. A* finds optimal path between any two nodes.
2. All nodes are connected using these minimal paths.
3. a complete helper graph is made with only Primary nodes.
4. Kruskal's algorithm finds the MST of the helper graph

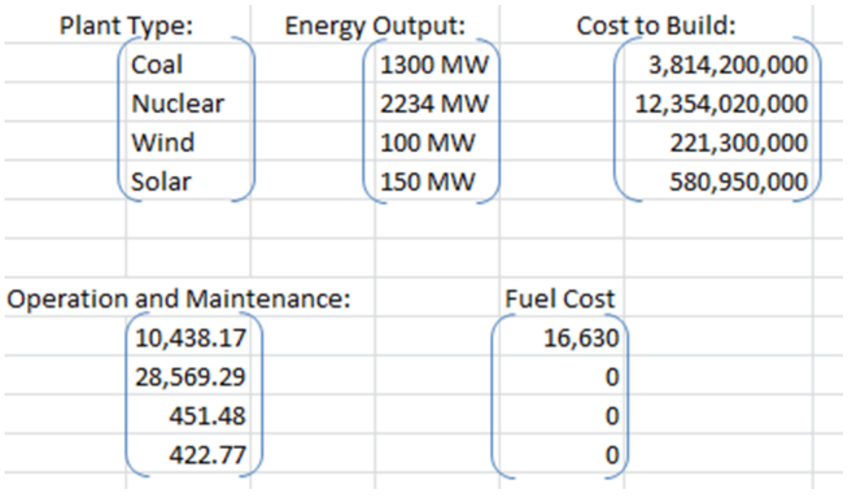
5. The actual cable routes are back-traced from the helper.

Production Strategy (Plant Type Selection)

The objective of this subsystem is to provide for the energy needs of a given geographic region, within a certain budget, and to do so in the most cost-effective manner. Given plant-specific costs, this subsystem makes an initial guess for providing the cheapest energy and is updated with additional costs from other subsystems to solve for a more accurate solution.

For the purposes of the plant type optimization, it was determined to minimize the cost per unit energy. It is important to factor in all known costs to determine what the most optimal plant array is. The costs associated with a given power plant include the cost to build the plant, purchase fuel for the plant, operate and maintain the plant, connect the power plant to the grid, and a cost associated with the "goodness" of a given location. This goodness penalty is an additional cost to penalize for placement in a socially unacceptable location or for placing a plant in a location that lacks adequate resources to achieve maximal energy output . All of these costs are tabulated for each iteration and an optimal plant array is determined with the most currently known information. As this subsystem gains more information throughout the optimization process, the cabling costs and the goodness penalty are further updated with information at the current timestep and new solutions are found until a global solution on the grid is determined.

The information utilized for plant type selection is shown in figure 5 below. It should be noted from this information that nuclear does actually have a fuel cost associated with it but the way information is collected, nuclear fuel costs are considered under operation costs. For the purposes of uniformity, energy output, operation and maintenance, and fuel costs are tabulated on a per hour basis.



Plant Type:	Energy Output:	Cost to Build:
Coal	1300 MW	3,814,200,000
Nuclear	2234 MW	12,354,020,000
Wind	100 MW	221,300,000
Solar	150 MW	580,950,000

Operation and Maintenance:	Fuel Cost
10,438.17	16,630
28,569.29	0
451.48	0
422.77	0

Figure 5. Plant specific energy output, build, and maintenance cost.

In regards to calculations performed for the optimization of this subsystem, it was determined fairly accurate and simple to calculate everything assuming that each power plant is in operation for 20 hours per day, 365 days per year. Research yielded that this is the industry standard for both coal and nuclear plants and although it is an overstatement for wind and solar, they are more variable in regards to location for how much energy can be extracted. The goodness penalty accounts for deviations in production for solar and wind plant types by penalizing areas where either sun or wind are not as available of resources and a monetary penalty is associated with decreased performance.

The objective function that is calculated by this subsystem is shown below, where ‘n’ equals the number of plants in the array, ‘P’ is the cost to build a plant, ‘F’ represents the fuel costs for a plant, ‘U’ represents the fixed and variable operation and maintenance costs, ‘C’ represents the cabling costs to link a power plant to the grid, ‘G’ represents the goodness cost for a given location, ‘E’ is the energy produced by a plant, ‘D’ is the energy demanded by the geographic region chosen, and ‘B’ is the budget allocated for the project.

$$min \sum_{i=0}^{n} \frac{P \bullet x_i + F \bullet x_i + U \bullet x_i + C(x_i) + G(x_i)}{E}$$

$$\text{s.t.} \quad g_1 = D - E \bullet x \leq 0$$

$$g_2 = P \bullet x + F \bullet x + U \bullet x + C(x) + G(x) - B \leq 0$$

This minimization problem was chosen because it yields an accurate solution in a simple manner.

Integration of Subsystems

The first step of the optimization scheme is to make an initial guess. For this project, the initial guess is determined on plant specific costs only and assumes zero cabling costs and no goodness penalty. This initial guess is in the form of a vector, called ‘x’ in the above plant type selection minimization problem, that specifies the type and number of plants that are found most optimal initially. The initial guess is sent to the location function, which determines how to best orient all of the plants to utilize resources in the best manner. Once the optimal locations for ‘x’ are known, the information is sent to the cabling function, which determines the minimum cost associated with connecting all of the power plants in ‘x’ to the grid. The information for goodness and cabling cost of the initial guess are then sent back to the plant type function, which updates the known costs and resolves for a more accurate ‘x’. This process continues until a global solution is found.

Optimization Study

Initially the plant location study was done in a brute force manner. Due to the small number of points in the mesh, 400, it was extremely cheap computationally to run through every point on every step to determine the best remaining locations for each plant type and number. Although this method was extremely fast and useful initially to help in the creation of other portions of the code, ultimately it did not use the optimization principles being taught in the class. Additionally, it would not have been able to handle larger meshes very well.

Once the final model was created for the location subsystem several optimization schemes were used including simulated annealing, Gradient Descent, and basinhopping. The only scheme to converge was the genetic algorithm scheme, which takes a population of possible solution candidates and with the use of a fitness function determines the best possible then it draws from that new pool and continues this as an iterative process. It is patterned after an evolutionary cycle.

Results

Parametric Study: Normalized Plant Size

For this simulation, all plant sizes were scaled to produce 10 TWh equivalent, yearly. This was done along with a change in cost (both construction and maintenance), so that the linear constants defined in the Production Strategy section remained the same. The interpretation is that an investor would put as much money required into construction of a type of plant so as to produce precisely 10TWh. The demand was set to reflect a state-wide energy need of 50 TWh, with a budget of $30bn USD, to more dramatically show the placement of plants.

The solution was found to be as follows:

Plant Type	#
Coal	0
Nuclear	0
Wind	4
Solar	3

Based on the normalized cost of the plants at the time of construction, the leftover budget for the year will be $1.05bn USD, with a projected net energy surplus of 20 TWh. Neither of these are at precisely 0, though the budget is close enough to be considered an active constraint at this solution, given the scale of the plants and the integer numbers of plants. It is more efficient in this case to over-produce, and max out the budget, to get the most energy for your dollar. The plant locations, overlaid on elevation, sunshine, wind, and water data is shown in Figure 6.

It is important to note that, since the plants can be so large, solar and wind power dominate. Wind is located in the high-velocity corridor between Phoenix and Tucson, and the remainder of the energy need is provided by solar arrays to the south-west, rather than building more wind in the White mountains. No nuclear or coal is needed, since at such high production rates per grid space, the transmission cost outweighs the benefit of short-term energy solutions at the 40-year prediction threshold.

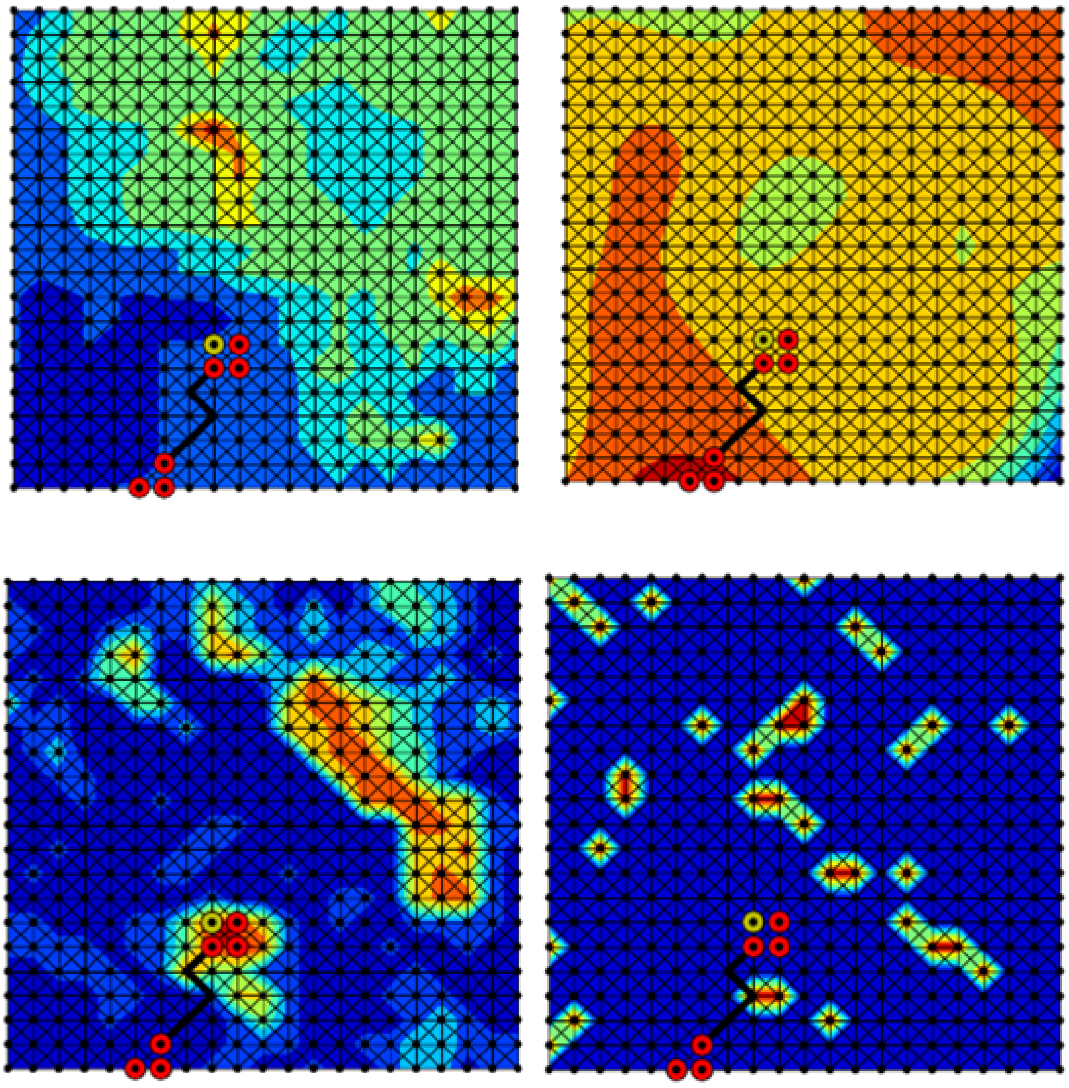


Figure 6. Plant locations for 10 TWh-normed production, on Elevation, Solar, Water Resources, and Wind data, CW from top-left.

Parametric Study: Realistic Plant Size

In the second simulation, plants retained their original power ratings, and the need was diminished to reflect Phoenix metropolitan area consumption (16.7 TWh yearly). The budget for this consideration was set to $9.5bn USD.

Plant Type	Production	#
Coal	9.36 TWh	1
Nuclear	16.08 TWh	0
Wind	720 GWh	8
Solar	1.08 TWh	2

The leftover budget in this case would therefore be $0.54bn USD, with an excess energy production of 640 MWh. The variation in production allows for a much closer fit to both needs, so neither are truly active constraints.

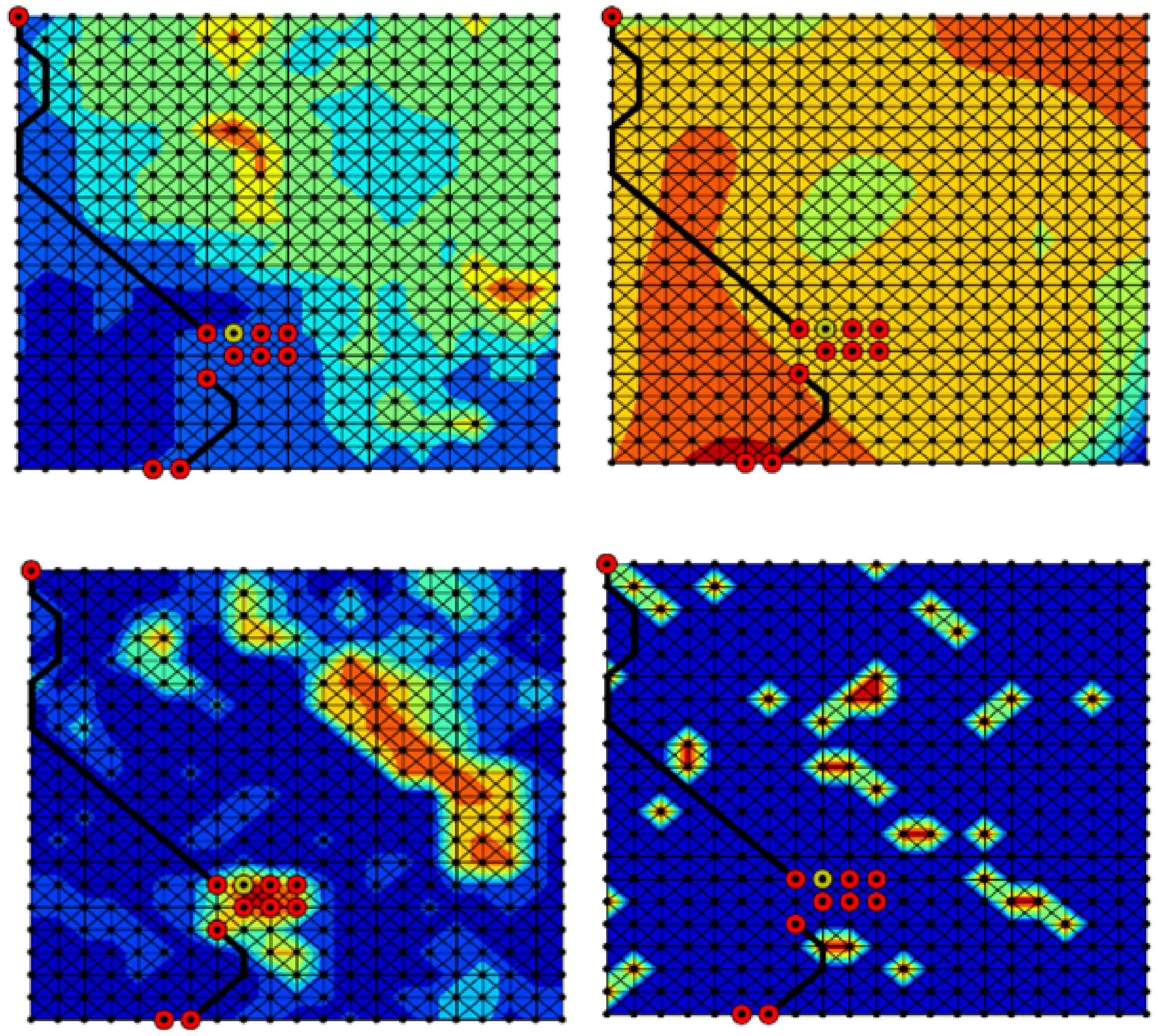


Figure 7. Plant locations for standard production, on Elevation, Solar, Water Resources, and Wind data, CW from top-left.

Even with the large transmission line distance, we see a huge portion of the need is supplied by a coal plant near the Nevada border, where transmission is not impeded by dramatic elevation changes Since the cash influx is more strained, the coal plant takes up the bulk of the need after all wind-spots nearby Phoenix are taken.

Conclusion

The optimization study performed was highly successful in showing that the concept works well. The group was able to perform two different studies, one with energy output normalized to ten Twh per year for each plant and another scaled to the outputs of the average power plant of a given type. Each of these resulted in fairly similar outcomes but due to the dramatically different output of a coal plant, when the outputs were not normalized, the results yielded differently. In the case of normalized energy output, the optimal plant array was found to contain four wind plants and 3 solar plants. When the average energy output data was utilized, the optimal plant array became one coal plant, eight wind plants, and two solar plants.

The optimization design utilized for this project determined results quickly and simply in a way that integrated all subsystems with decent weighting for each. If the project was taken further, the equations utilized for goodness may need to be modified to better suit the mentality of the individuals within the region of study and as the number of nodes analyzed increases, it may become necessary to further optimize the code to be less computationally expensive. Research performed led the group to choose the weightings that we chose and it is believed that all numbers chosen accurately represent the region chosen. Overall, for the purposes of this project, the group believes that the optimal solution has been accurately found and that the study was a success.

References

2014, February. "Energy Efficiency and Energy Consumption." ARIZONA ENERGY FACT SHEET (n.d.): n. pag. SWEEP. Web.

ABC of Water. (n.d.). Retrieved May 9, 2015, from http://www.azwater.gov/AzDWR/PublicInformationOfficer/ABCofWater.htm

Arizona Renewable Energy Resource Maps. (n.d.). Retrieved May 9, 2015, from http://apps1.eere.energy.gov/states/maps.cfm/state=AZ

Clean Energy Research and Education. (n.d.). Retrieved May 9, 2015, from http://nau.edu/CEFNS/Engineering/Mechanical/Research-and-Labs/Energy/Energy/Wind-Resourc e-Maps/

Logistic function. (n.d.). Retrieved May 9, 2015, from http://en.wikipedia.org/wiki/Logistic_function

TopoCreator. (n.d.). Retrieved May 9, 2015, from http://topocreator.com/download_city_a.php

"U.S. Energy Information Administration - EIA - Independent Statistics and Analysis." U.S. Energy Information Administration (EIA). N.p., n.d. Web. 10 May 2015.

Appendix

```
"""
Location Finding Subsystem Script
Created on Mon Apr 27 11:01:59 2015

@author: Thurston
"""


import numpy as np
from scipy.interpolate import RectBivariateSpline
from scipy.special import expit
import scipy.stats

elev =np.flipud(np.genfromtxt('elev.txt', delimiter=','))/12615.
water=np.flipud(np.genfromtxt('water.txt'))
wind =np.flipud(np.genfromtxt('wind.txt', delimiter=','))

sun_scat =np.flipud(np.genfromtxt('sun.txt', delimiter='\t'))

x,y = (np.arange(0,21), np.arange(0,21))
X,Y = np.meshgrid(x,y)

from scipy.interpolate import Rbf
rbf = Rbf(sun_scat[:,0], sun_scat[:,1], sun_scat[:,2])
#f=scipy.interpolate.interp2d(sun_scat[:,0], sun_scat[:,1], sun_scat[:,2], kind='linear')
sun = rbf(X, Y)


def dist_func(x, y):
    return (np.sqrt((8-x)**2+(6-y)**2)*15)/277.

def coal_log(x, y):
    d = np.sqrt((8-x)**2+(6-y)**2)*15
    #print d
    d_new = (6/277.)*(d - 138.5)
    #print d_new
    return 1-expit(d_new)

def nuke_log(x, y):
    d = np.sqrt((8-x)**2+(6-y)**2)*15
    #print d
    d_new = (4/15.)*(d - 45)
    #print d_new
    return 1-expit(d_new)

shape,loc,scale = scipy.stats.lognorm.fit(wind.flatten(), floc=0)
wind_norm=scipy.stats.lognorm.cdf(wind, shape, loc, scale)
overlap=np.zeros_like(X)

def goodness(plant_type, overlap):
    if plant_type=='wind':
        good_mat= overlap+0.7*(1-wind_norm)+.3*dist_func(X,Y)

    elif plant_type=='coal':
        good_mat = overlap+.5*coal_log(X,Y)+.3*dist_func(X,Y)-.2*water

    elif plant_type=='nuke':
        good_mat = overlap + .5*nuke_log(X,Y)+.3*dist_func(X,Y)-.2*water

    elif plant_type=='solar':
        good_mat = overlap + 0.6*(1-sun) + 0.4*dist_func(X,Y)

    return good_mat

def find_locs(plants):
    print 'finding locs'
    plant_types = ['coal','nuke','wind','solar']
    locs=[]
    overlap=np.zeros_like(X)
    #print 10e9-np.multiply(x0, E).dot(K)
    #print np.dot(x0, E) - 160.55e9

    bnds = ((0, 20), (0, 20))

    for n,j in enumerate(plants):
        plant_type=plant_types[n]
        def how_good(u):
            u=[round(i) for i in u]
            #print u, RectBivariateSpline(x, y, goodness(plant_type)).ev(u[0],u[1])
            return RectBivariateSpline(x, y, goodness(plant_type, overlap)).ev(u[0],u[1])
        for i in range(int(j)):
            #print n, i
            res = scipy.optimize.differential_evolution(how_good, bounds=bnds)
            #print res.fun
            loc=np.flipud(res.x.round())

            overlap[int(loc[1]), int(loc[0])]+=1000.
            #print overlap
            locs.append((loc[0],loc[1]))
    return locs
```

```
"""
Minimum Spanning Tree Subsystem script
Created on Mon Apr 27 10:54:29 2015

@author: Thurston
"""

import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import itertools as it

x,y = (np.arange(0,21), np.arange(0,21))

X,Y = np.meshgrid(x,y)

list1 = []
for i in range(21):
    for j in range(21):
        list1 += [(x[i], y[j])]

node_pos = {i:list1[i] for i in range(441)}

elev = np.flipud(np.genfromtxt('elev.txt', delimiter=','))/12615.
water= np.flipud(np.genfromtxt('water.txt'))
wind = np.flipud(np.genfromtxt('wind.txt', delimiter=','))
sun_scat =np.flipud(np.genfromtxt('sun.txt', delimiter='\t'))
from scipy.interpolate import Rbf
rbf = Rbf(sun_scat[:,0], sun_scat[:,1], sun_scat[:,2])
#f=scipy.interpolate.interp2d(sun_scat[:,0], sun_scat[:,1], sun_scat[:,2], kind='linear')
sun = rbf(X, Y)

dX, dY = np.gradient(elev)
dElev = np.sqrt(dX**2) + np.sqrt(dY**2)


G=nx.empty_graph(0)
G.name="grid_diags_graph"
rows=x
columns=y

G.add_nodes_from((i,j) for i in rows for j in columns)

node_pos = {list1[i]:list1[i] for i in range(441)}
for n, p in node_pos.iteritems():
    G.node[n]['pos'] = p
    G.node[n]['elev'] = elev[n[0],n[1]]
    G.node[n]['water'] = water[n[0],n[1]]
    G.node[n]['wind'] = wind[n[0],n[1]]

G.add_edges_from( ((i,j),(i-1,j),
                   {'weight':1+abs(G.node[(i, j)]['elev']-G.node[(i-1, j)]['elev'])+\
                   .3*G.node[(i, j)]['water']*G.node[(i-1, j)]['water']}) for i in x for j in y if
i>0 )
G.add_edges_from( ((i,j),(i,j-1),
                   {'weight':1+abs(G.node[(i, j)]['elev']-G.node[(i, j-1)]['elev'])+\
                   .3*G.node[(i, j)]['water']*G.node[(i, j-1)]['water']}) for i in x for j in y if
j>0 )
G.add_edges_from( ((i,j),(i-1,j-1),
                   {'weight':np.sqrt(2)+abs(G.node[(i, j)]['elev']-G.node[(i-1,
j-1)]['elev'])/np.sqrt(2)+\
                   .3*G.node[(i, j)]['water']*G.node[(i-1, j-1)]['water']}) for i in x for j in y
if (j>0 and i>0) )
G.add_edges_from( ((i,j),(i+1,j-1),
                   {'weight':np.sqrt(2)+abs(G.node[(i, j)]['elev']-G.node[(i+1,
j-1)]['elev'])/np.sqrt(2)+\
                   .3*G.node[(i, j)]['water']*G.node[(i+1, j-1)]['water']}) for i in x for j in y
if (j>0 and i<20) )




def min_span_tree(G, plot=False, *plant_locs):
    print 'finding MST'
    T = nx.empty_graph(0)

    T.add_node((8,6), {'wii':'Phoenix'})
    all_locs = [(8,6)]+[i for i in plant_locs]
    T.add_nodes_from(plant_locs)

    edges=it.combinations(all_locs,2)
    #print edges
    T.add_edges_from((i[0],i[1],
                     {'weight':nx.shortest_path_length(G,
                                                       source=i[0],
                                                       target=i[1],
                                                       weight='weight')}) for i in list(edges))

    #print T.edges()
    #print nx.edges_iter(T)

    M=nx.minimum_spanning_tree(T)
    M.graph['name']='Optimal Grid Transmission Lines'

    #nx.draw(M,node_pos)
    plot_edges = []
    path=[]
    for i in M.edges_iter():
        #print i
        path += nx.shortest_path(G,source=i[0],target=i[1])
        #print path[-1], path[1:]
    plot_edges = [x for x in zip(path,path[1:]) if not (x in T.edges()) and not ((x[1],x[0]) in
T.edges())]
    #plot_edges= zip(path,path[1:])
    #plot_edges= list(set(zip(path,path[1:])) - set(T.edges()))
    #print plot_edges
    if plot==True:
        f,ax = plt.subplots(2,2, figsize=(9,30))
        ax[0][0].contourf(X,Y,elev)
        nx.draw_networkx_nodes(M,node_pos,node_size=200, node_color='r', ax=ax[0][0])
        nx.draw_networkx_nodes(M,node_pos,node_size=200,nodelist=[(8, 6)], node_color='y',
ax=ax[0][0])
        nx.draw(G, node_pos, node_size=20, with_labels=False, node_color='k', ax=ax[0][0])
        nx.draw_networkx_edges(G,node_pos,edgelist=plot_edges,edge_color='k',width=5, ax=ax[0][0])

        ax[1][0].contourf(X,Y,water)
        nx.draw_networkx_nodes(M,node_pos,node_size=200, node_color='r', ax=ax[1][0])
        nx.draw_networkx_nodes(M,node_pos,node_size=200,nodelist=[(8, 6)], node_color='y',
ax=ax[1][0])
        nx.draw(G, node_pos, node_size=20, with_labels=False, node_color='k', ax=ax[1][0])
        nx.draw_networkx_edges(G,node_pos,edgelist=plot_edges,edge_color='k',width=5, ax=ax[1][0])

        ax[0][1].contourf(X,Y,wind)
        nx.draw_networkx_nodes(M,node_pos,node_size=200, node_color='r', ax=ax[0][1])
        nx.draw_networkx_nodes(M,node_pos,node_size=200,nodelist=[(8, 6)], node_color='y', 
ax=ax[0][1])
        nx.draw(G, node_pos, node_size=20, with_labels=False, node_color='k', ax=ax[0][1])
        nx.draw_networkx_edges(G,node_pos,edgelist=plot_edges,edge_color='k',width=5, ax=ax[0][1])

        ax[1][1].contourf(X,Y,sun)
        nx.draw_networkx_nodes(M,node_pos,node_size=200, node_color='r', ax=ax[1][1])
        nx.draw_networkx_nodes(M,node_pos,node_size=200,nodelist=[(8, 6)], node_color='y', 
ax=ax[1][1])
        nx.draw(G, node_pos, node_size=20, with_labels=False, node_color='k', ax=ax[1][1])
        nx.draw_networkx_edges(G,node_pos,edgelist=plot_edges,edge_color='k',width=5, ax=ax[1][1])

    #print plot_edges
    #mst=nx.minimum_spanning_edges(T,data=True) # a generator of MST edges
    #edgelist=list(mst) # make a list of the edges
    #print(sorted(edgelist))
    #print ((8,6),(5,13)) in T.edges()
    #print 'Positions: ', T.nodes()
    #print nx.info(M)
    #print plot_edges[-1]
    #print M.size(weight='weight')
    return M.size(weight='weight')
```

```
###########
Ipython Notebook for Top lvl. (Plant production strategy)
###########

import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import itertools as it
from scipy.optimize import minimize
import scipy
#import seaborn as sns

%matplotlib

# 6 types of plants:
# coal, nuclear, wind, solar

Y = 40 #predictive years
mu = 0.05 #initial capital extra from projected upkeep (ratio)

#(build cost + (upkeep/yr)*years*mu)/Energy output(kWh)

E = [9.36e9, 16.0848e9, 720e6, 1.08e9]
#E = [9.36e9, 16.0848e9, 1.4e9, 5.08e9]

k_coal = (3.8142e9 + 7.516e7*Y*mu)
k_nuke = (12.35e9 + 2.06e8*Y*mu)
k_wind = (221e6 + 3.3e6*Y*mu)
k_solar = (580.95e6 + 3.04e6*Y*mu)

K = np.divide([k_coal, k_nuke, k_wind, k_solar],E)

print K

E_min = 50e9/3. #kWh
C_max = 30e9/4. #USD

%run locs.py
%run mst.py


def plant_combo(x):
    x=[round(i) for i in x]

    plant_locs=find_locs(x)
    extra=min_span_tree(G, *plant_locs)

    obj=np.dot(x, K) + extra*15*1e6/np.dot(x, E)

    '''if np.dot(x, E) - E_min >=0: g1 = 0
    else: g1=20
    if C_max-np.multiply(x, E).dot(K)>=0: g2 = 0
    else: g2=20
    #print obj+g1+g2
    return obj+g1+g2'''
    #print C_max-np.multiply(x, 10e9).dot(K) #cost net
    #print np.dot(x, 4*[10e9]) - E_min
    return obj


poss = [int(E_min/E[0])+1, int(E_min/E[1])+1, int(E_min/E[2])+1, int(E_min/E[3])+1]
print poss[0]+1
```

```
x_test=it.product(range(poss[0]), range(poss[1]), range(poss[2]), range(poss[3]))

plant_combs=list(x_test)
print len(plant_combs)
#print plant_combs

test_vals=[]
x_store=np.empty((1,4))

for i in plant_combs:
    #print i
    x = [float(j) for j in i]
    if (np.dot(x, E) - E_min>=0)and( C_max-np.multiply(x, E).dot(K)>=0):
        print i
        test_vals+=[plant_combo(x)]
        x_store=np.vstack((x_store, x))

f,ax = plt.subplots(1,4, figsize=(12,3))
ax[0].plot(x_store[1:,0], test_vals, 'bo')
ax[1].plot(x_store[1:,1], test_vals, 'bo')
ax[2].plot(x_store[1:,2], test_vals, 'bo')
ax[3].plot(x_store[1:,3], test_vals, 'bo')

print x_store[np.argmin(test_vals[1:])].tolist()
print C_max-np.multiply(x_store[np.argmin(test_vals[1:])].tolist(), E).dot(K)
print np.dot(x_store[np.argmin(test_vals[1:])].tolist(), E) - E_min
print find_locs(x_store[np.argmin(test_vals[1:])].tolist())
min_span_tree(G,True, *find_locs(x_store[np.argmin(test_vals[1:])].tolist()))
```