Meeting Notes (Bashar, 10/22)

Project 1 (high priority):

- Objective function(x,y), inequality constraints, equality constraints
- X can be continuous (R), y can be discrete (Z)
- Includes linear and nonlinear functions
- Convert non linear functions to piecewise linear approximations (convert to MILP)
- Work: How to formulate piecewise linear functions (multivariate)
- Operations Research (OR) literature has models such as multiple choice, convex combination, disaggregated log, etc…)
- GDP: Generalized Disjuncted Programming
- Questions: Can we use GDPs to generate piecewise models through transforms?
- Set up a server to run, run the GDP script on different versions of Gurobi (8, 9, 10), run different analyses to see which model wins in garb every year (may change across different years)
- Tools: Code script (provided), Server (need access), Install diff versions of Gurobi
- Goal: Run script on each version, then run analysis script on results
- First Step: Read through papers (next 2 weeks), create ppt slides reviewing things learned from papers (general overview, what I understood/didn't understand, questions), review GDP (when book comes in)
- Note: Each run can take around a month, runs in the background (3 runs total)
- Note: Papers are mostly bivariate or univariate, but we will be doing multivariate analyses

Project 2 (low priority):

- GDP Overview: Say there are some constraints to apply to a model if decision X is made, a different set of constraints apply if decisions Y is made. Can systematically apply transformations that convert these constraints into MIPs (convex hull, big m, multiple big m)
- Hull has the highest relaxation strength (typically for bigger models), big M is tight (smaller models), MBM is in the middle (in between models), but have to calculate for multiple big M (high up front computational cost)
- Convect combination model is a parallelizable MBM computation, is there a way to code this up in pyomo to calculate multiple Big M in parallel?

Meeting Notes (Bashar, 11/06):

- Action Plan: Create a fork of the OMLT Repo, clone the fork, start contributing to OMLT
- Next Step: Revise slide deck with specific things that I've learned
- cog-imperial/OMLT
- Find and submit old OMLT Pyomo Homework from last semester

Meeting Notes (Bashar, 11/13):

- Attempt to recreate error from issue 165 in cog-imperial/OMLT, in a new .py file
- See what I can do to debug this error and/or trace it back (short term, immediate project)
- Review future of trees in OMLT
- Goal: Unify everything under a common tree class, update to support all types of inputs
- What are the important attributes of a decision tree to accommodate all outputs needed
- Combine LightGBM, Linear Tree, Sci-kit learn
- Create intermediate TreeDefinition class that combines all three input types
- Output types will then take from treeDefinition formulations
- Write an API that transforms decision trees in whatever tool into TreeDefinition class
- People should be able to then add based on the treeDefinition class.
- What are the key attributes of a decision tree to include? How do we want to build the class? What methods? What is cache vs calculated?
- Ex: Previously, y_index is calculated, but it has been stored in cache. Children should also be moved from cache to calculated.
- Install firefox (JSON reader), can be downloaded
- Can open files directly in firefox, use dropdowns to access sub-dictionaries
- Look at lightGBM data structure, dump it as a JSON, from there get all the information to write the tree.
- To start, generate fake data and train a basic model in LightGBM. Train a tree on fake data, multi-input and multi-output. Input and output sizes do not have to be the same. Generate 1000 data points (input output pairs). Train a standard-tree model with constant leaves, and then a linear tree (linear leaves) (leaves = data type of nodes, either constant for standard or some y=mx+b for linear). Return that model as a JSON, identify the key information in the tree.
- From this, now see how this can relate to the tree definition class.
- First work on issue 165, then train the new tree. Issue should be within the next 2-3 days (send updates by friday).
- Use np.random for the dataset, we don't care about this model. 4-d sin wave?

Python Multiprocessing Notes (11/17):

- Package that allows for parallelism
- Strong ties to functional programming (map, folding, etc…)
- Processes are spawned by creating a "Process" object and calling start() method
- Installation: pip install multiprocessing
- Import: from multiprocessing import Process
- 3 ways to start: Spawn, Fork, Forkserver
- MacOS defaults to spawn, familiarize mostly with spawn as origin
- Uses an API similar to threading module (also for parallelism)
- Process communication channels: Queues and Pipes
- Queues serialize objects, and operate in a single direction
- Pipes are duplex by default and communicate both ways, with send() and recv()

- send() serializes an object, recv() re-creates a serialized object on the other end.
- Manager() object controls a server process holding several objects
- Manager objects allow other processes to manipulate objects by proxy
- Managers are slower than using shared memory, but more flexible in supporting arbitrary types
- Pool class represents a pool of worker objects, allows tasks to be offloaded in parallel to worker processes, but the method of a pool must be used only by the process that created it.
- x.start() starts process x, x.join() waits until process x has terminated before continuing
- X.is_alive returns true if X is still running, false otherwise
- os.getpid() is how to obtain the process ID when defining a worker
- Lock() ensures that only one process is executing a code section at a time
- Processes run independently and each have their own memory space
- Array and Value objects can be used to share data between processes
- Syntax: https://www.geeksforgeeks.org/multiprocessing-python-set-2/?ref=next_article
- List and Dict methods can be used in the Manager() class
- Pipe data can be corrupted if two processes/threads write to the same end simultaneously
- Queues are inherently safer and will not be corrupted (thread-safe)
- Each process is an instance of the byte-code python interpreter (lower than user-level)
- Overall Guide: https://superfastpython.com/multiprocessing-in-python/

Meeting Notes (Bashar, 11/20):

- Cached: things that are inherent to the tree, parents, intercepts, slopes, values
- Computed: knowing what is cached, we can traverse the tree and calculate the bounds
- Remove bounds from cached, that should be computed
- Tree Definition:
    - Self.leaves = {tree{leaf{'parent': nodeID, 'slopes': vector inputs, 'intercept': value}
    - This is what we get when we initialize (attributes)
    - compute_bounds(self) (fxn that returns data structure with bounds, single leaf or all)
    - When someone returns tree def class, only thing they will have is self.leaves
    - When we go to actually implement, we will then have TreeDefinition.compute_bounds
    - Gives user more insight into the tree models, no need to extract data from the tree definition class making it more versatile
- Train models, dump to JSON, JSON will only include things that must be cached. From there model the linear trees integration, see what is common between JSON and implementation, create a tree definition class that can handle what all has been trained.
- Regression, Classification, categorical, linear, constant trees, multi output
- Open locally saved file in firefox through file->open
- Type exit() after every run of the python script since it stays in python otherwise and will give syntax errors.
- Try uninstalling and reinstalling VS code to fix python exit issue.
- In pyomo there is a GDP transformation into MIPs, one of those methods is multiple big m (MBM), calculates the enumerated disjuncts, want to parralelize calculating the enumerations 10x9 -> 1x9

- This calculation specifically happens in pyomo->gdp->plugins->MBM transform-> calculate_missing_M_values has a for loop (for all disjunct and other disjunct), it builds LP/Solve LP in gurobi, want to parallelize that for loop since LPs are independent.
- Most likely batch them, want user to be able to set batch number (limited by number of gurobi licenses), need n licences for n batch processes.
- Sent pyomo interface via email (for MBM transform parallelization, line 595 for loop)

Train many LGBM models

- Multi input single output
- Single input multi output
- Categorical inputs
- Continuous inputs/outputs
- Single Tree/Multi tree
- Random forest/gradient boosted tree
- All permutations of the above
- Explore how all this information is captured in the JSON files
- Get as many JSON files as possible
- Find a single data structure that encompasses all the permutations
- NOTE: Only applies to cached data
- Put everything into one concise powerpoint by end of week
- Throw all JSONs in, remove .txt files
- Throw everything into a github repo, along with powerpoints
- Pick out snippets that define each case, along with the corresponding overall data structure
- Summarize in a powerpoint with snippets