Mixed Integer Non-Linear Programming; An Exploration

0.1 Introduction & Problem Definition

Suppose the following problem, introduced by Dr. Wilson in the "Operations Research; Application and Algorithms" textbook as a primer to Non-Linear Programming:

- *If K units of capital and L units of labor are used, a company can produce KL units of a manufactured good. Capital can be purchased at \$4/unit and labor can be purchased at \$1/unit. A total of \$8 is available to purchase capital and labor. How can the firm maximize the quantity of the good that can be manufactured?* [9]

We can formalize the problem mathematically, such that

$$\begin{aligned} \max z &= KL \\ s.t. \quad & 4K + L \leq 8 \\ & K, L \geq 0 \end{aligned}$$

In the original problem formulation, K and L are treated as continuous variables, capable of assuming any value in the real number field so long as the defined constraints are met. This is an assumption that predicates the rest of the Non-Linear Programming chapter, but one that is not necessarily valid for many real life examples. For instance, capital is only as "continuous" as the smallest existing denomination of it. Recognizing the discrete nature of certain variables is integral in accurately modelling real life optimization problems. In particular, the consideration of *integers* is vital for the effective optimization of systems where discrete decisions are made. Such systems are particularly common in engineering models, most prominent in the modelling of discrete decisions on complex chemical engineering processes, as well as economic and transportation/network flow models [8]. As this paper will later explore, they also appear in the optimization of sports games targeted primarily to children under the age of 18.

Mixed Integer Non-Linear Programming (MINLP) is the study of non-linear systems where certain variables assume integer values. A Mixed Integer Non-Linear Program can be most abstractly formalized as the following:

$$\begin{array}{lll} \min & \mathrm{z = f(x,\ y)} & \forall x \in X, y \in Y \\ \text{s.t.} & \mathrm{g}_k(x, y) \leq 0 & \forall k \leq l \\ & \mathrm{X} \subseteq \mathbf{R}^n, & \\ & \mathrm{Y} \subseteq \mathbf{Z}^m & \end{array}$$

Verbalized, this mathematical formulation states that we are attempting to minimize a function f defined by n real variables and m integer variables, given l constraint functions g.

Computationally, MINLPs have proven to be particularly challenging optimization problems, both on an algorithmic and complexity front. Relaxations of the MINLP optimization class to either Mixed-Integer Linear Programs (MILPs) or Non-Linear Programs (NLPs) produce $\mathcal{NP}$-hard problems in both cases. As such, combining both problems to form MINLPs implies that some such problems are indeed undecidable [7]. Despite this, there exists a class of MNILPs that are 'easier' to solve and thus more thoroughly researched, namely *convex* MINLPs. In this chapter, we consider and contrast state-of-the-art algorithms for both convex and non-convex MINLPs, before exploring computational methods for solving MINLPs.

0.2 Convex Mixed Integer Non-Linear Programs

We note that a function $f : \mathbf{R}^n \to \mathbf{R}$ is considered convex if for any x, y in the domain and $\lambda \in [0, 1]$,

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Graphically, we can understand this as stating f applied to any two points between x, y lies under the line segment created between $f(x)$ and $f(y)$.

Rigorously, in the context of formal definitions of convexity, Mixed Integer Non-Linear Programs are inherently non-convex, due to the presence of discrete variables. Albeit such, we define the class of convex MINLPs to be problems where the program would be considered convex were discreteness conditions relaxed (i.e. if we interpret the discrete variables as continuous), effectively assessing the convexity of the sister NLP [7]. An NLP is considered convex if the minimization objective function and upper-bounded constraint functions are all convex functions [2].

The late mathematician R. Tyrrell Rockafellar was quoted in saying *"...in fact, the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity"*. Convex MINLP problems are a particularly relevant subclass to consider insofar as their relationship to Mixed Integer Linear Programming. Specifically, one can interpret the *linearization* property oresent to convex functions as the affirmation of the relationship between linear and convex non-linear problems [2].

Linearization is the process of determining a linear approximation to a given function f at a certain point. Given the assumption of a convex objective function, the point of optimality inherently occurs within the convex hull of some set of feasible solutions [2]. As such, we can reconstruct the MINLP with an "auxiliary variable" δ, such that:

$$\begin{array}{lll} \min & \delta & \\ \text{s.t.} & f\ (\mathrm{x}_1, \cdots, x_n) \leq \delta & \\ & g_k\ (\mathrm{x}_1, \cdots x_n) \leq 0 & \forall k \leq l \\ & \mathrm{x}_1, \cdots, x_i \in \mathbf{Z}, & \\ & \mathrm{x}_{i+1} \cdots x_n \in \mathbf{R}. & \end{array}$$

From here, we consider the fact that f and g_k are assumed to be convex and differentiable. As such, at any given point $\bar{x}$, the function in question will remain above its gradient linear tangent, such that

$$f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}) \leq f(x) \leq \delta$$
$$g_k(\bar{x}) + \nabla g(\bar{x})^T (x - \bar{x}) \leq g_k(x) \leq 0$$

Where ∇ is the partial derivative operator. Specifically, the linearization of these functions in this manner is known as a *gradient cut linear relaxations*. We should note that gradient cut linearization presents a lower bound for the objective function $f(x)$, while creating an upper bound for the constraints $g_k(x)$ [2]. These are leveraged in certain iterative *decomposition* methods further explored below, such that as the monotonically increasing lower bound converges to the upper bound integer constraints, we converge to optimality [1]. In effect, we can fundamentally understand and visualize gradient cut linearization as a tool that allows us to create linear outer bounds around the convex function. By strategically adding more gradient cut linearizations, we iteratively approach the optimal solution. Figure 1 presents a two-dimensional graphical representation of this, where we see that linearization create a linear problem space, and the addition of more gradient cut linearizations visibly narrows it.

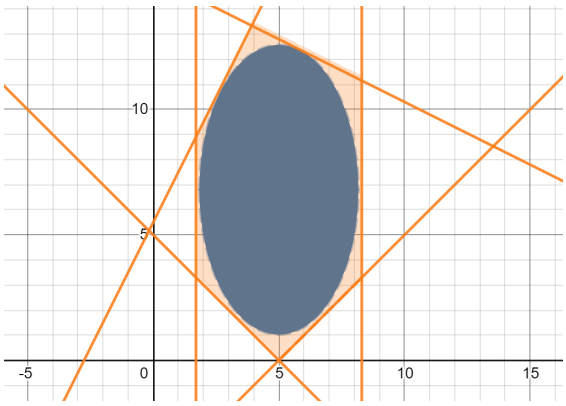

Figure 1: Graphical Representation of gradient cut linear relaxations

The question then becomes - how do we add the gradient cuts in a "strategic" way to find the optimum? By extension, do we need to employ linearizations at all to solve MINLPs? These are question many convex MINLP algorithms attempt to solve. We first present a familiar algorithm as a means of solving MINLPs. It is worthwhile noting that some solutions and algorithms developed for convex MINLP problems are actually iterative adaptations of MILP algorithms. One such example is the branch and bound method:

0.2.1 The Branch & Bound Method applied to MINLPs

From as early as 1965, Dakin recognized that the branch and bound method commonly used for solving MILPs did not necessitate linearity, stating that non-linear programs could also consider this class of solutions (limitations of his time forbade him from exploring this further) [3]. What he did not explicitly specify was that the assumed linearity condition at the time was actually a weaker condition of convexity [3]. This opened the door for research in the application of branch-and-bound to MINLPs [2]. In 1985, Gupta and Ravindran presented a formal exploration of the branch and bound method applied to MINLPs [2].

In their paper, Gupta and Ravindran assert the condition of "first order sufficient optimality" on the sister NLP of the convex MINLP in question [6]. This simply states that some KKT point is indeed the global minimum of the sister NLP. Given this, they present a branch and bound algorithm for MINLPs almost identical to the standard MILP branch and bound, whereby:

1. Relax the integral constraints and solve for the accompanying NLP
2. If the solution is integral, then an optimal solution of the node has been achieved. Else, chose a continuous variable x_i, and compartment it into an integer $[x_i]$ and decimal $\hat{x_i} \in [0, 1]$, such that $x_i = [x_i] + \hat{x_i}$. From this, consider two branch instances of the MINLP, where one has the added constraint $x_i \leq [x_i]$, and the other has the added constraint $x_i \geq [x_i] + 1$.
3. Repeat (1) and (2), branching further and further until a node solution is infeasible or an integral solution is achieved. At this point, the node has been "fathomed". Each fathomed node presents an upper bound of the optimal minimization solution.
4. Assess all fathomed nodes. The one with the smallest solution presents optimality [6].

Due to the familiarity and relative simplicity of the algorithm, the branch and bound method is primarily recommended when having to compute MINLPs by hand. Particularly, it is (relatively) easy to apply for smaller problems, and does not require the methodological processing of other algorithms. However, where branch and bound falters is with computational efficiency, more specifically dynamic computational efficacy. While other algorithms attempt to shrink the solution space dynamically, the branch and bound method requires that every node be visited before determining an optimal solution, effectively not making use of information garnered as we traverse the problem. Despite existing heuristic considerations and the use of "pseudo-costs" allowing us to determine which branches are more likely to produce the optimal solution, the branch and bound method necessitates that every branch be traversed. For the time being we keep it in the back of our minds as we consider another, dynamic approach to solving MINLPs.

0.2.2 The Outer Approximation MINLP Method

Just one year after Gupta & Ravindran introduced the Branch-and-Bound (B&B) algorithm to MINLPs, Duran & Grossman presented the *Outer Approximation* algorithm [5]. Similarly to the branch and bound (B&B) algorithm, the Outer Approximation (OA) algorithm is iterative. However, while the B&B algorithm iteratively decomposes a problem space until a solution (or no solution) is found for a marginal subset, the OA method *decomposes* the problem into NLP and MIP components through an algorithmic *relaxation* approach. In iteratively solving these 'easier' sub-problems, optimality is reached.

In order to understand what a relaxation algorithm is, we first definethe term. A *relaxation* of a supposed set U is any set V such that $U \subseteq V$ [5]. Considering this, relaxation algorithms attempt to iteratively create tighter relaxation sets that converge to optimality. In order to create a valid relaxation, the entire original set must be contained within the relaxation set. Graphically, this can translate to the relaxation space never 'cutting' the original space. The original Outer Approximation algorithm introduced by Duran & Grossman specifically considers linearized relaxations of the MINLP in question, iteratively using gradient cuts to tighten the relaxation set until we converge to the optimal solution. More formally, we can define the Duran & Grossman Outer Approxation algorithm as follows [5]:

1. Relax the integral condition of an MINLP to solve it as an NLP
2. Add a gradient cut linearization at the NLP point of optimality.
3. Consider the relaxed set created by the added linearization. Seeing as the relaxation cut is linear, we can find its point of optimality with LP algorithms, or more specifically MIP algorithms. As such, solve the 'master MIP' within the scope of the relaxed set.
4. If the MIP solution is infeasible, revert back to step 1, but with an added integer constraint (i.e, the integer value found is assumed true for the NLP as a constant, but not considered for the next MIP iteration). If the MIP solution is feasible, then you have achieved the optimal MINLP solution.

The last clause is predicated on the intuitive notion that if the minimum (or maximum) of a relaxation set V is within the bounds of the set U, then it must also be the minimum (or maximum) of the set U, or else $\min U \notin V$ (this is a contradiction, as then $U \nsubseteq V$, which cannot be the case by the definition of V as a relaxation). Keeping this in mind, we recognize that the OA method is a fairly intuitive and simple algorithm, that relaxes the problem in a way that lets consider simpler problem subclasses. We now present a very simple example problem to further visualize the OA algorithm:

$$\begin{aligned} \min z = \quad & 0.5x + y \\ \text{s.t.} \quad & (x-1)^2 + y^2 \leq 3 \\ & x \in [0, 3], \\ & y \in \{0, 1, 2\}. \end{aligned}$$

Solving this as an NLP, we find that approximately, $x = 1.77$ and $y = 1.55$. We solve for the gradient cut linearization at this point by solving for the tangent at the optimal point, which produces the linear equation:

$$y = -0.5x + 2.43 \tag{1}$$

From here, we now solve for the MIP with the linear relaxation (1) (with a $\leq$ sign) as a constraint instead of the non-linear original constraint. We find an optimal solution at roughly $(2.879, 1)$, and find that this is indeed an infeasible solution for the problem. At this point, we now solve for the NLP with the original constraints, as well as the added constraint that $y = 1$. We solve to find an optimum at $x = 2.414$ and $y = 1$. We once again find the gradient at this point, which equates to

$$y = -1.41x + 4.41 \tag{2}$$

We now once again return to solving the MIP, this time with the added constraints that $y \neq 1$, and the added constraint from (2) (again with a $\leq$ sign). We find a new master MIP optimal solution at $(0.864, 2)$. However, at this point, we run into a problem - there is no solution in the original NLP where $y = 2$! At this point, we simply remove $y = 2$ from the scope of solutions, just as we had done for $y = 1$ prior. The difference is that we did not accompany this with an added linear constraint, effectively missing out on the opportunity to tighten the bound.

With more modern iterations of the algorithm, infeasible NLP solutions are not ignored, but rather a 'closest' solution is found to the infeasible solution. This ensures that the linear relaxations are being strengthened with every iteration, and improves overall computational efficiency of the algorithm. Some very recent algorithms have adapted this further to not only consider the closest point to the infeasible solution, but

rather the closest point to the 'least infeasible' solution. Essentially, this means that given the solution $(0.864, 2)$ is infeasible, the least infeasible solution will be found (i.e. a solution on the line $y = 2$ that is closest to the feasible space). And then from there, the algorithm will find the closest point to that least infeasible solution. This creates a tighter bound, further strengthening the relaxation set. In a sense, these are optimizations on top of optimizations of optimization algorithm - at this point, one must ask where the line is to be drawn!

Humour aside, we return to the MIP, now with the added constraint that $y = 2$ is not a feasible solution either. We solve the MIP to find a solution at $(3, 0)$. Going back to the NLP, we find the respective solution at $x = 2.73$, $y = 0$. We add the gradient cut constraint $x \leq 2.73$ to the MIP. Removing $y = 0$ from the scope of solutions for the MIP, we find that we have exhausted all possible solutions. As such, we assess all the feasible solutions we have found, to conclude that $(2.879, 1)$ is the optimal one. Figure 2 visualizes the OA process we went through for this problem:

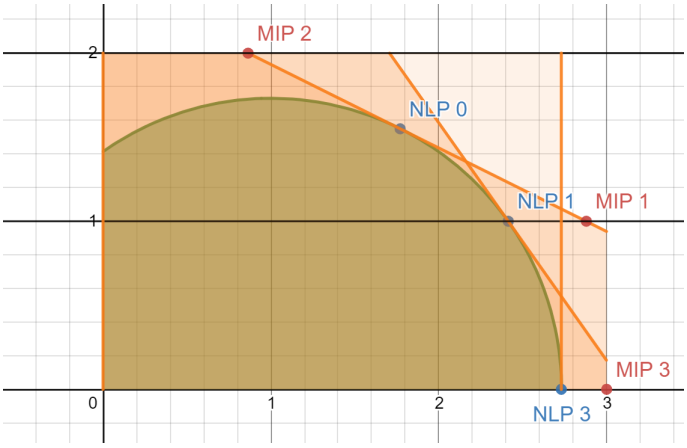

Figure 2: Visualization of the Outer Approximation method

This particular problem was chosen as a demonstration of the worst case scenario for the OA algorithm. Had one of the MIP solutions been feasible, then the algorithm would have terminated. Nevertheless, we show the feasibility of the OA algorithm, which allows us to apply algorithms and processes already familiar to us through the power of linear relaxations.

Comparing the Outer Approximation algorithm to the Branch-and-Bound algorithm, we find that both aim to decompose the convex space in some capacity so as to have to consider simpler sub-classes of optimization problems, namely NLP and MIPs. The OA method on smaller problems is indeed more challenging, as we have to not only construct the linear bounds, but also solve for both NLPs and MIPs (as opposed to the just solving for NLPs and segmenting accordingly with B&B). Having said this, the OA algorithm is special in that it boasts a global termination case, whereas the B&B only presents a local one. What is meant by this is that once a solution is found for the MIP that is feasible to the MINLP, then the algorithm immediately terminates. In the case of B&B, finding a feasible solution only fathoms a single node, which then has to be compared to its sibling node in a recursive process. Heuristically, we can suppose that the OA algorithm will likely iterate through less iterations than the B&B [7]. However, this is keeping in mind the fact that one iteration of the OA is computationally more demanding than one iteration of the B&B. As such, the different solutions exhibit differences that make one or the other more suitable for different questions. We recommend the Outer Approximation algorithm for larger problems, where the computational cost saved from strengthening the bounds will likely far outweigh the constant time added computational cost to each iteration. Conversely, with smaller problems, the constant overheads attached to OA will likely make it less computationally favorable than B&B, especially considering the fact that there are only so many branches that can be created with smalller problems.

0.3 Non-Convex MINLPs

Having explored a couple of methods for solving convex MINLPs, we now consider the much challenging class of non-convex MINLPs, where the ability to create global bounds with linear relaxations is no longer applicable. More formally, with non-convex MINLPs, we cannot assume that a gradient cut remains as a lower bound in the case of the objective function, or an upper bound in the case of constraints. As such, we cannot assume that the set created by linearized gradient cuts is a relaxation of the non-convex problem space. This effectively nullifies both proposed convex MINLP algorithms. In the case of Branch-and-Bound, the terminating condition of a node being 'fathomed' if an integral solution is achieved is no longer valid, as it was the convexity condition that ensured no further branching down the path would produce a more optimal solution. Similarly yet more drastically, the Outer-Approximation algorithm completely falters without the supposition of convexity, as the linear gradient cuts generated by definition no longer necessarily create relaxation spaces of the problem. As such, the termination condition no longer holds - a minimum solution in the linearized space that is feasible no longer implies minimimality in the feasible space.

The study of non-convex MINLP solvers is still very much active, as solutions remain incredibly complex, computationally challenging, and limited to certain subclasses of non-convex MINLPs [7]. In this section, we present a high level overview of an existing method for solving a class of non-convex MINLP

0.3.1 Convex Relaxations of Non-convex MINLPs

Similarly to how convex MINLP algorithms such as OA attempted to simplify the scope of the problem by relaxing the problem space to be linear, we try to solve non-convex MINLPs by considering convex relaxations of it. Unfortunately, this is easier said than done.

A *convex hull* of some non-convex set U is defined to be the smallest convex set V such that $U \subseteq V$ [7]. This can also be interpreted as the tightest convex relaxation of the set U. The issue, however, is that finding convex hulls is not very easy. In fact, we simply do not know how to find the convex hull of all non-convex sets - this remains an unsolved problem [7]. Despite this, we do know how to construct convex relaxations for some simpler non-convex sets. As such, generally, when solving non-convex MINLPs, we attempt to simplify and segment the problem as much as possible, so that we can find convex relaxations [7]. One such way to do this is to consider individual non-convex constraints, and try to find convex relaxations for them [7]. We then consider a convex relaxation of the entire set to be the intersection of the convex relaxations of the constraints [7]. Formally, we can write this as attempting to determine, for every non-convex constraint g_k whereby we note (by the definition of MINLPs) that $g_k(x) \leq 0$ for all elements in the domain of the MINLP, a convex function g'_k, such that

$$g'_k(x) \leq g_k(x) \leq 0.$$

Sometimes, this may be a simple task, but that may not always be the case. At such a point, we attempt to consider the individual elements of g_k that are composed together in some way to form g_k, and consider their convex sets. Supposing that we are successful in at least finding convex relaxations of individual parts of the constraint function in question g_k, we can compose them together to form *piece-wise* convex or concave envelopes around the non-convex set using what is known as *McCormick Envelopes* [7]. The McCormick envelop, also known as the McCormick relaxation, is a complex transformation function that takes in the convex counterpart of the non-convex components of a constraint, and then transforms them in such a way where the produced function is a sufficiently tight' convex lower bound [4].

At this point, we consider the different piece-wise convex functions constructed as per the constraints, and then solve for them using the methods explored above, amongst other methods developed for convex MINLP - we could even go on step further and relax the convex relaxations with linear relaxations! We note that the solutions obtained by solving the piece-wise convex MINLPs are lower bounds, and as such we aim to construct convex relaxations that are as tight as possible [4].

0.4 Solving MINLPs with Pyomo

Despite our unwavering esteem and admiration for LINGO, we consider a new computational tool of choice - Pyomo. Pyomo, which stands for Python Optimization Modelling Objects, is a framework for constructing and defining optimization problems. Pyomo is not itself a solver, but rather a framework through which optimization problems are defined for solvers to effectively interpret. Adopting a modelling framework such as Pyomo ensures that our model construction is formalized and consistent, which is essential when working with large programs. In particular, we adopt Pyomo to solve the 'No Links Wasted' FIFA Ultimate Team problem previously explored. As a quick recap, we consider the following specifications, reconstructing the problem within the context of an MINLP:

Figure 3: Team Network Structure & Constant Players

Given a large (750) database of players, we seek to construct a team of eleven players (three of which are constants) such that the rating of the entire team, defined to be the sum of the ratings of the players, is maximized. Players are constrained by their ascribed position, as well as the chemistry links they share with other players. For a player to be included in the team, the following conditions must be met:

- The player is in the right position within the formation.
- Within the network of the team, the player has a cumulative non-negative set of chemistry links. With the added No Links Wasted stipulation, each player must have a net zero set of chemistry links.
  - Recall that a chemistry link between two players can assume values of -1, 0, or 1, depending on if players linked in the graph share the same nationality and/or league.
  - The value of the chemistry link between two players defines the added value from that link for both players. For example, the green link between Gundogan and Werner in the team graph implies that both players are currently on a +1 net chemistry value. This implies that there other chemitry links must have a net value of -1.

Considering these conditions, we formalize relevant variables and functions for the construction of a MINLP.

- Let x_i be a binary variable denoting the inclusion of player i in the team.
- Consider the suite of functions required for constructing the MINLP:
  - Let $r : \mathbf{P} \to \mathbf{N}$ be the function that given a player $i \in P$, returns the player's rating.

- Let $pos : \mathbf{P} \rightarrow \mathbf{Pos}$ be the function that given a player $i \in \mathbf{P}$, returns the players position. Note that $\mathbf{Pos} = \{\text{ST, LM, CM, RM, LB, CB, RB, GK}\}$
- Let $link : \mathbf{P} \times \mathbf{P} \rightarrow \{0, 1\}$ be the function that given players $i, j \in \mathbf{P} \times \mathbf{P}$, returns 1 if there exists a link from the position of player i to the position of player j in the formation network.
- Let $chem : \mathbf{P} \times \mathbf{P} \rightarrow \{-1, 0, 1\}$ be the function that given players $i, j \in \mathbf{P} \times \mathbf{P}$, returns -1 if the players share neither a nation nor a league, 0 if the players share either a nation or a league, and 1 if the players share both.

Having defined our terms, we now construct our MINLP. We first confirm our objective function:

$$\min z = -\sum_{i \in \mathbf{P}} r(i)x_i$$

As constraints, we require position as well as chemistry constraints. For position constraints, for every position $p \in \mathbf{Pos}$, we define a constraint stating that the sum of all $x_i \in \mathbf{P}$ such that $pos(x_i) = p$ must equate to 1. To formalize this, we consider a function $pbool : \mathbf{Pos} \times \mathbf{Pos} \rightarrow \{0, 1\}$, where if two positions inputted are the same, it returns 1, returning 0 otherwise.

$$\sum_{i \in \mathbf{P}} pbool(pos(i), p) \times x_i = 1, \forall p \in \mathbf{Pos}/\{ST, CM, CB\}$$

These constraints ensure that exactly 1 player in every position is chosen. Given that Gundogan, Werner, and Hummels are all to be set as constants (1), the constraints ascribed to their positions (ST, CM, CB) will have an equality bound of 2.

$$\sum_{i \in \mathbf{P}} pbool(pos(i), p) \times x_i = 2, \forall p \in \{ST, CM, CB\}$$

Moving onto the chemistry constraint, we want to define constraints such that for all players in the optimal solution, the chemistry links they have with all the players they can share links with equate to 0. This effectively emulates the NLW chemistry constraint. We can formalize this set of constraints as follows:

$$x_j \times \sum_{i \in \mathbf{P}} link(i, j) \times chem(i, j) \times x_i = 0, \forall j \in \mathbf{P}/w, g, h$$

The multiplication of each summation by x_j effectively allows us to 'turn off' the constraints for players that do not end up appearing in the optimal solution. This is necessary, as we now avoid instances where a player's chemistry and position constraints affects optimal players when he should not be in the solution.

Having defined the problem scope, we now implement the problem structure on Pyomo. We first define our variables, before adding the objective function, followed by the relevant constraints. A noteworthy remark to make is that all any data being filtered or considered in Pyomo needs to be converted to the respective Pyomo data structure. While this is indeed a tedious process, it ensures consistency as Pyomo structures the data for the relevant solver, in this case the 'ipopt' solver from the 'MindtPy' solver package. This is demonstrated in the accommodating code package to this folder, which presents the integration of the problem into Pyomo.

The MindtPy solver extension, which stands for Mixed-Integer Nonlinear Decomposition Toolbox, supports convex MINLP solving, and thus by extension all the simpler sub-classes (NLPs, MILPs, LPs) [1]. MindtPy specifically considers and presents the option to use different decomposition algorithms, where gradient cut linearizations are iteratively considered as they converge to an optimal solution from a lower bound [1]. MindtPy is considered state-of-the-art for Pyomo integrated solvers primarily due to the suite of decomposition algorithms available [1]. MindtPy presents intergrations for the Outer Approximation method developed above, as well as advanced variations of it for different specific convex MINLP problem subclasses, such as the

Generalized Bender Decompositions method, and the Partial Surrogate Cuts method [1]. In this instance, we use the default MINLP OA solver from MindtPy.

Having constructed the problem and employed a convex MINLP solver, we garner the following:

```
In [225]: print('Chosen Players:')
          for i in model.N:
                  if(pe.value(model.X[i])> 0.99):
                      print(data[i - 1][0])
          print()
          print('Objective Value:', pe.value(model.obj))
```

```
Chosen Players:
Karim Benzema
Mats Hummels
Yann Sommer
Timo Werner
Serge Gnabry
Lucas Digne
Kingsley Coman
Ilkay Gündogan
Nabil Fekir
Theo Hernandez
Santiago Arias

Objective Value: 928.0006204665538
```

Figure 4: MintPy Output

Visualizing this into a team, we produce the team below. This does indeed seem to be a No Links Wasted team, and it should be optimal. In the previous report, I included, rather confidently at that, what I thought the optimal team would have been, and this team actually exceeds that one (with a rating of 928 against my optimal team rating of 927). I guess in conclusion, years of FIFA playing still looses out to a computer. But hey, at least I now have a truly *ultimate* FIFA Ultimate Team.

Figure 5: Optimal Team Visualized

.

References

[1] Bernal, David E., Qi Chen, Felicity Gong, and Ignacio E. Grossmann. "Mixed-Integer Nonlinear Decomposition Toolbox for Pyomo (MindtPy)." In Computer Aided Chemical Engineering, edited by Mario R. Eden, Marianthi G. Ierapetritou, and Gavin P. Towler, 44:895–900. 13 International Symposium on Process Systems Engineering (PSE 2018). Elsevier, 2018. https://doi.org/10.1016/B978-0-444-64241-7.50144-0.

[2] Bonami, Pierre, Mustafa Kilinç, and Jeff Linderoth. Algorithms and Software for Convex Mixed Integer Nonlinear Programs, 2009. https://hal.archives-ouvertes.fr/hal-00423416.

[3] Dakin, R. J. "A Tree-Search Algorithm for Mixed Integer Programming Problems." The Computer Journal 8, no. 3 (January 1, 1965): 250–55. https://doi.org/10.1093/comjnl/8.3.250.

[4] Dombrowski, John. "McCormick Envelopes - Optimization." Northwestern Mathematics Wiki. Accessed May 6, 2021. https://optimization.mccormick.northwestern.edu/index.php/McCormick_envelopes

[5] Duran, Marco A., and Ignacio E. Grossmann. "An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs." Mathematical Programming 36, no. 3 (October 1, 1986): 307–39. https://doi.org/10.1007/BF02592064.

[6] Gupta, Omprakash K., and A. Ravindran. "Branch and Bound Experiments in Convex Nonlinear Integer Programming." Management Science 31, no. 12 (1985): 1533–46.

[7] Mixed Integer Nonlinear Programming, edited by Jon Lee and Sven Leyffer, 1–39. The IMA Volumes in Mathematics and Its Applications. New York, NY: Springer, 2012. https://doi.org/10.1007/978-1-4614-1927-3_1.

[8] Sahinidis, Nikolaos V. "Mixed-Integer Nonlinear Programming 2018." Optimization and Engineering 20, no. 2 (June 1, 2019): 301–6. https://doi.org/10.1007/s11081-019-09438-1.

[9] Winston, Wayne L. Operations Research: Applications and Algorithms. PWS-Kent Publishing Company, 1991.