**Exploring the Gurobi Solver**

**Introduction**

Mathematical optimization is a critical component in solving complex decision-making problems across a multitude of industries. The Gurobi Optimizer is one of the leading tools in the field of Solver software, and is renowned for its robustness, efficiency, and wide applicability. This blog post will provide a look at the solver, starting with historical development, an explanation of what a solver is, how Gurobi specifically works, and delving into the applications and a use case with Gurobi.

**History of Gurobi**

Gurobi was developed by Dr. Zonghau Gu, Dr. Edward Rothberg, and Dr. Robert Bixby in 2008 as part of their LLC, Gurobi Optimization. The name Gurobi comes from the first two letters in each of their surnames – **Gu** **Ro**thberg, and **Bi**xby. The founders were prominent figures in the optimization community, with Dr. Bixby being one of the co-founders of CPLEX, which is another well-known solver. Gurobi was designed to provide a more modern, high-performance solver that leverages the latest and greatest in computer science and mathematical optimization. Over the years, Gurobi has outperformed other solvers in various benchmarks, and has grown its reputation as a top choice for optimization applications.

**What is a Solver?**

Before discussing more about Gurobi specifically, it is important to address what a solver is, and what it is meant to do. A solver is a software tool designed to find the best solution to a mathematical optimization problem. Optimization problems involve the maximization or minimization of an objective function bounded by a set of constraints. Solvers are essential in a number of fields, including operations, finance, logistics, and engineering, where it is crucial to ensure that the solution is within the boundaries of the constraints.

**Famous Optimization Problems**

To shed some more light on what an optimization problem looks like, here are some famous examples that have influenced the field of optimization and algorithm development:

* **Traveling Salesman Problem**: Given a set of cities and the distances between them, the Traveling Salesman problem seeks to find the shortest route to visit each city once before returning to the origin city. This is a classic combinatorial optimization problem and is a blueprint for logistics and network design solutions. An illustrated solution for this problem in the context of the contiguous United States is below.

A map of the united states

Description automatically generated

Figure 1: An illustration of the solution to the Traveling Salesman problem as applied to the contiguous United States.

* **The Knapsack Problem**: In this problem, a set of items with a weight and value must be selected to maximize the value of the knapsack while keeping the weight of the bag below a weight capacity threshold. This is an example of an inventory management and resource allocation problem.
* **The Assignment Problem**: This problem involves assigning a set of tasks to a number of agents to minimize the total cost and time. This has applications in job scheduling, resource allocation, and personnel assignment.
* **The Vehicle Routing Problem**: The vehicle routing problem is concerned with optimizing a fleet of vehicles to service a set of customers while minimizing the cost and distance traveled by the vehicles. This is a case of a logistics and transportation planning problem and is reminiscent of how delivery services are optimized.

**How Gurobi Works**

Now that the background of what Gurobi is and some of the problems it can solve has been laid out, we can now explore what exactly makes Gurobi, Gurobi. Gurobi uses modern algorithms and techniques to solve a variety of optimization problems. The solver employs a combination of methods, such as the simplex algorithm, interior point methods, and branch-and-bound techniques to efficiently navigate the solution space and identify the most optimized ones. To expand on those:

* **Simplex Algorithm**: Developed by George Dantzig in 1947, the simplex algorithm is widely used for solving linear programming problems. The constraints of the problem define the vertices of the feasible region, which is then operated on by the algorithm. It iteratively moves from one vertex to the next in a way that the objective function value improves at each step. This continues until either an optimal solution is found or that a feasible solution does not exist.
* **Interior-Point Methods**: An alternate to the simplex algorithm to solve linear and nonlinear programming problems, interior-point methods work by traversing the interior of the feasible region rather than the boundary. First developed by Narendra Karmakar in 1984, these methods make use of advanced techniques such as barrier functions and Newton’s method to find an optimal solution. These are effective for large-scale problems and often outperform the simplex algorithm in these cases.
* **Branch-and-Bound Techniques**: This is a general algorithmic method of finding optimal solutions to a variety of combinatorial optimization problems. It involves dividing the original problem into subproblems (branching) then solving these subproblems systematically. By calculating the bound of the best possible solution of each subproblem, the algorithm can eliminate subproblems that cannot yield a solution better than the current best (bounding), until the optimal solution is found.

**Applications of Gurobi**

Gurobi Optimizer is a flexible tool that can be used to solve a broad spectrum of complex problems for an equally broad spectrum of industries. Here’s a look into a few use cases, along with some sample constraints:

* **Supply Chain Optimization**: A retail company wants to minimize logistic costs while ensuring its warehouses are stocked per customer demand. Gurobi can be used to create a model to minimize those transportation costs while satisfying production capabilities and customer demand constraints.
* **Finance**: A financial analyst is creating a portfolio for a customer with a certain capacity to invest with a maximal allowable risk capacity. Gurobi can be used to identify the optimal asset mix that can maximize the portfolio performance while adhering to the risk profile of the customer and the amount that the customer is willing to invest, while managing the regulatory constraints.
* **Energy and Utilities**: An energy provider is looking to optimize its power generation schedule to minimize cost of production while meeting the demand of its customers and adhering to environmental emissions limits. Gurobi can help identify the best schedule for generators to manage power distribution while minimizing emissions.

**Practical Use Case**

To illustrate the use of Gurobi in practice, let’s consider a simple linear programming problem from the supply chain management industry. Suppose we want to maximize the profit of producing two products, A and B. Each product requires a certain amount of resources to produce, and there is a constraint on the total amount of resources available to the company. First, let’s rewrite this to clarify the problem.

**Objective:**

* Maximize profit P
  + PA, the profit of product A: $20/unit
  + PB, the profit of product B: $30/unit

**Resource Constraints**

* Resource 1: The company has 100 units of resource 1.
  + Product A requires 2 units to produce
  + Product B requires 1 unit to produce.
* Resource 2: The company has 80 units of resource 2.
  + Product A requires 1 unit to produce
  + Product B requires 2 units.

Now that we’ve formulated this into a problem, we can see the mathematical portion start to show itself as a series of equations.

We could solve this by hand through solving the system of equations from the constraints, but it could take several iterations to determine the most optimal solution, by solving for the max profit of only producing A, and only producing B, and then iteratively increasing and decreasing the values of A and B to find the maximum profit. But since we don’t want to do that, we can use the following Julia code to do this with Gurobi instead

**Example Julia Code**

using Gurobi

using JuMP

## Create a Gurobi Model

gu\_mod = Model(Gurobi.Optimizer)

## Create variables

@variable(gu\_mod, A >= 0)

@variable(gu\_mod, B >= 0)

## Create Constraints

@constraint(gu\_mod, 2\*A + B <= 100)

@constraint(gu\_mod, A + 2\*B <= 80)

## Set the objective

@objective(gu\_mod, Max, 20\*A + 30\*B)

## Optimize the Model

optimize!(gu\_mod)

## Print the results

println("Optimal Values:\n------------------")

println("Product A: ", value(A))

println("Product B: ", value(B))

println("Maximum Profit: ", objective\_value(gu\_mod))

Running this with a Gurobi license results in the production values of A = 40 and B = 20, for a maximum profit of $1400. These values of A and B also optimize the constraints. To solve this without a license, similar code was used in Python using the CVXPY library, which has been included in an appendix along with the direct terminal output.

**Conclusion:**

To conclude, Gurobi is a powerful optimization solver that can be applied to a variety of industries to solve a broad set of complex optimization problems. Gurobi uses modern computing techniques to solve these problems, including the simplex algorithm, inner-point methods, and branch-and-bound techniques. This blog explored some industry examples where Gurobi can be applied and provided a sample practical solution method to solve a linear optimization problem. Additionally, a few famous optimizations were explored to give context on optimization problems that have influenced the field of optimization. Ultimately, as long as optimization continues to play a critical role in many industries, solver tools like Gurobi will continue to be indispensable.

**Appendix**

1. Python code to solve the supply chain practical use-case

import numpy as np

import cvxpy as cp

## Define variables

a = cp.Variable()

b = cp.Variable()

## Define constraints

con1 = 2\*a + b <= 100

con2 = a + 2\*b <= 80

constraints = [con1, con2, a >= 0, b >= 0]

## Define objective

objective = cp.Maximize(20\*a + 30\*b)

## Solve

prob = cp.Problem(objective, constraints)

prob.solve()

print("Status: ",prob.status)

print("Optimal Value: ", np.round(prob.value))

print("a: ", np.round(a.value))

print("b: ", np.round(b.value))

1. Terminal Output from the Python Code

Status: optimal

Optimal Value: 1400.0

a: 40.0

b: 20.0

**References**

[1] R. E. Bixby, "A brief history of linear and mixed-integer programming computation," Documenta Mathematica, vol. Extra Volume ISMP, pp. 107-121, 2012.

[2] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com/documentation/9.5/refman/index.html. [Accessed: 25-Jul-2024].

[3] E. Rothberg, "An Introduction to Gurobi," Gurobi Optimization, 2024. [Online]. Available: https://www.gurobi.com/resource/an-introduction-to-gurobi/. [Accessed: 25-Jul-2024].

[4] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in *Activity Analysis of Production and Allocation*, T. C. Koopmans, Ed., New York: Wiley, 1951, pp. 339-347.

[5] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373-395, 1984.

[6] D. P. Bertsekas, "Nonlinear Programming," 2nd ed., Belmont, MA: Athena Scientific, 1999.

[7] P. van Hirtum, "An Overview of the Traveling Salesman Problem," *Journal of Optimization Theory and Applications*, vol. 138, no. 3, pp. 569-593, 2008.