Tutorial IV.IV - Advanced Solver Options with HiGHS in JuMP

Applied Optimization with Julia

Introduction

Welcome to this tutorial on advanced solver options in JuMP using the HiGHS solver! Don't worry if "advanced solver options" sounds intimidating - we'll break everything down into simple, easy-to-understand concepts.

Imagine you're using a GPS app to find the best route to a new restaurant. Just like how you can adjust settings in your GPS (like avoiding toll roads or preferring highways), we can adjust settings in our optimization solver to help it find solutions more efficiently or to meet specific requirements.

By the end of this tutorial, you'll be able to: 1. Understand what solver options are and why they're useful 2. Set basic solver options like time limits and solution tolerances 3. Interpret solver output to understand how well your problem was solved

Let's start by loading the necessary packages:

```julia
using JuMP, HiGHS
```

Section 1: Understanding Solver Options

Solver options are like the "advanced settings" of our optimization tool. They allow us to control how the solver approaches our problem. Here are a few common options:

1. Time limit: How long the solver should try before giving up
2. Solution tolerance: How precise we need the answer to be
3. Presolve: Whether to simplify the problem before solving it

Let's create a model and set some of these options:

```julia
model = Model(HiGHS.Optimizer)

# Set a time limit of 60 seconds
set_time_limit_sec(model, 60)

# Set the relative MIP gap tolerance to 1%
set_optimizer_attribute(model, "mip_rel_gap", 0.01)

# Turn on presolve
set_optimizer_attribute(model, "presolve", "on")
```

```
println("Solver options set successfully!")
```

```
Solver options set successfully!
```

Let's break this down:

- `set_time_limit_sec(model, 60)` tells the solver to stop after 60 seconds if it hasn't found a solution
- `set_optimizer_attribute(model, "mip_rel_gap", 0.01)` sets how close to the best possible solution we need to be (within 1%)
- `set_optimizer_attribute(model, "presolve", "on")` tells the solver to try simplifying the problem first

Exercise 1.1 - Set Solver Options

Now it's your turn! Set the following solver options: 1. A time limit of 120 seconds 2. A MIP gap tolerance of 0.5% 3. Turn off presolve

```
# YOUR CODE BELOW
```

```
# Test your answer
@assert time_limit_sec(model) == 120 "The time limit should be 120 seconds
but is $(time_limit_sec(model))"
@assert solver_name(model) == "HiGHS" "The solver should be HiGHS but is
$(solver_name(model))"
@assert MOI.get(model, MOI.RawOptimizerAttribute("mip_rel_gap")) == 0.005
"The MIP gap should be 0.5% but is $(MOI.get(model,
MOI.RawOptimizerAttribute("mip_rel_gap")))"
@assert MOI.get(model, MOI.RawOptimizerAttribute("presolve")) == "off"
"Presolve should be off but is $(MOI.get(model,
MOI.RawOptimizerAttribute("presolve")))"
println("Great job! You've successfully set advanced solver options.")
```

Section 2: Creating and Solving a Sample Problem

To see how these options affect solving, let's create a simple optimization problem. We'll use a basic production planning scenario.

Imagine you're managing a small factory that produces two types of products: widgets and gadgets. You want to maximize profit while staying within your production capacity.

```
# Define variables
@variable(model, widgets >= 0, Int)
@variable(model, gadgets >= 0, Int)

# Define constraints
@constraint(model,
```

```
    production_time,
    2*widgets + 3*gadgets <= 240
)
@constraint(model,
    widget_demand,
    widgets <= 80
)
@constraint(model,
    gadget_demand,
    gadgets <= 60
)

# Define objective (profit)
@objective(model,
    Max,
    25*widgets + 30*gadgets
)

# Solve the problem
optimize!(model)

# Print results
println("Optimization status: ", termination_status(model))
println("Objective value: ", objective_value(model))
println("Widgets to produce: ", value(widgets))
println("Gadgets to produce: ", value(gadgets))
```

```
Running HiGHS 1.11.0 (git hash: 364c83a51e): Copyright (c) 2025 HiGHS under
MIT licence terms
MIP  has 3 rows; 2 cols; 4 nonzeros; 2 integer variables (0 binary)
Coefficient ranges:
  Matrix [1e+00, 3e+00]
  Cost   [2e+01, 3e+01]
  Bound  [0e+00, 0e+00]
  RHS    [6e+01, 2e+02]
Presolving model
1 rows, 2 cols, 2 nonzeros  0s
1 rows, 2 cols, 2 nonzeros  0s
Objective function is integral with scale 0.2

Solving MIP model with:
   1 rows
   2 cols (0 binary, 2 integer, 0 implied int., 0 continuous, 0 domain
fixed)
   2 nonzeros

Src: B => Branching; C => Central rounding; F => Feasibility pump; J =>
Feasibility jump;
     H => Heuristic; L => Sub-MIP; P => Empty MIP; R => Randomized
rounding; Z => ZI Round;
     I => Shifting; S => Solve LP; T => Evaluate node; U => Unbounded; X =>
User solution;
     z => Trivial zero; l => Trivial lower; u => Trivial upper; p =>
```

```
Trivial point

        Nodes      |    B&B Tree     |            Objective Bounds
|  Dynamic Constraints |       Work
Src  Proc. InQueue |  Leaves   Expl. | BestBound       BestSol
Gap |   Cuts   InLp Confl. | LpIters     Time

 J       0       0         0   0.00%   inf             2000
Large        0      0      0         0     0.0s
 S       0       0         0   0.00%   3800            2780
36.69%        0      0      0         0     0.0s
         1       0         1 100.00%   2800            2780
0.72%        0      0      0         1     0.0s

Solving report
  Status            Optimal
  Primal bound      2780
  Dual bound        2800
  Gap               0.719% (tolerance: 1%)
  P-D integral      0.000312955782702
  Solution status   feasible
                    2780 (objective)
                    0 (bound viol.)
                    0 (int. viol.)
                    0 (row viol.)
  Timing            0.01 (total)
                    0.00 (presolve)
                    0.00 (solve)
                    0.00 (postsolve)
  Max sub-MIP depth 0
  Nodes             1
  Repair LPs        0 (0 feasible; 0 iterations)
  LP iterations     1 (total)
                    0 (strong br.)
                    0 (separation)
                    0 (heuristics)
Optimization status: OPTIMAL
Objective value: 2780.0
Widgets to produce: 80.0
Gadgets to produce: 26.0
```

This problem determines how many widgets and gadgets to produce to maximize profit, given time constraints and maximum demand.

Exercise 2.1 - Modify and Solve the Problem

Now it's your turn! Modify the problem above by:

1. Changing the production time constraint to 300 minutes
2. Increasing the profit for widgets to 30
3. Solving the modified problem and printing the results

```julia
# YOUR CODE BELOW
# Hint: Copy the code above and make the necessary changes
model = Model(HiGHS.Optimizer) # Don't forget to re-initialize the model
```

```julia
# Test your answer
@assert termination_status(model) == MOI.OPTIMAL "The termination status should be OPTIMAL but is $(termination_status(model))"
@assert isapprox(objective_value(model), 3780, atol=1e-6) "The objective value should be 3780 but is $(objective_value(model))"
@assert isapprox(value(widgets), 80, atol=1e-6) "The number of widgets to produce should be 80 but is $(value(widgets))"
@assert isapprox(value(gadgets), 46, atol=1e-6) "The number of gadgets to produce should be 46 but is $(value(gadgets))"
println("Excellent work! You've successfully modified and solved the optimization problem.")
```

Section 3: Interpreting Solver Output

When we solve an optimization problem, the solver gives us information about how it went. Let's look at some key pieces of information:

```julia
println("Termination status: ", termination_status(model))
println("Primal status: ", primal_status(model))
println("Dual status: ", dual_status(model))
println("Objective value: ", objective_value(model))
println("Solve time: ", solve_time(model))
```

Let's break this down:

- Termination status: Tells if the solver found an optimal solution, ran out of time, etc.
- Primal status: Indicates if we have a valid solution for our original problem
- Dual status: Relates to the mathematical properties of the solution (don't worry too much about this)
- Objective value: The value of our objective function (in this case, our profit)
- Solve time: How long it took to solve the problem

Conclusion

Well done! You've completed the tutorial on advanced solver options with HiGHS in JuMP. You've learned how to set advanced solver options. Continue to the next file to learn more.

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Bibliography