From MATLAb help

Tuning Integer Linear Programming

Change Options to Improve the Solution Process

After you run `intlinprog` once, you might want to change some options and rerun it. The changes you might want to see include:

- Lower run time
- Lower final objective function value (a better solution)
- Smaller final gap
- More or different feasible points

Here are general recommendations for option changes that are most likely to help the solution process. Try the suggestions in this order:

1. For a faster and more accurate solution, increase the `CutMaxIterations` option from its default `10` to a higher number such as `25`. This can speed up the solution, but can also slow it.
2. For a faster and more accurate solution, change the `CutGeneration` option to `'intermediate'` or `'advanced'`. This can speed up the solution, but can use much more memory, and can slow the solution.
3. For a faster and more accurate solution, change the `IntegerPreprocess` option to `'advanced'`. This can have a large effect on the solution process, either beneficial or not.
4. For a faster and more accurate solution, change the `RootLPAlgorithm` option to `'primal-simplex'`. Usually this change is not beneficial, but occasionally it can be.
5. To try to find more or better feasible points, increase the `HeuristicsMaxNodes` option from its default `50` to a higher number such as `100`.
6. To try to find more or better feasible points, change the `Heuristics` option to either `'intermediate'` or `'advanced'`.
7. To try to find more or better feasible points, change the `BranchRule` option to `'strongpscost'` or, if that choice fails to improve the solution, `'maxpscost'`.
8. For a faster solution, increase the `ObjectiveImprovementThreshold` option from its default of zero to a positive value such as `1e-4`. However, this change can cause `intlinprog` to find fewer integer feasible points or a less accurate solution.
9. To attempt to stop the solver more quickly, change the `RelativeGapTolerance` option to a higher value than the default `1e-4`. Similarly, to attempt to obtain a more accurate answer, change the `RelativeGapTolerance` option to a lower value. These changes do not always improve results.

Some “Integer” Solutions Are Not Integers

Often, some supposedly integer-valued components of the solution `x(intcon)` are not precisely integers. `intlinprog` considers as integers all solution values within `IntegerTolerance` of an integer.

To round all supposed integers to be precisely integers, use the `round` function.

```
x(intcon) = round(x(intcon));
```

Caution

Rounding can cause solutions to become infeasible. Check feasibility after rounding:

```
max(A*x - b) % see if entries are not too positive, so have small infeasibility
```

```
max(abs(Aeq*x - beq)) % see if entries are near enough to zero
max(x - ub) % positive entries are violated bounds
max(lb - x) % positive entries are violated bounds
```

Large Components Not Integer Valued

`intlinprog` does not enforce that solution components be integer valued when their absolute values exceed `2.1e9`. When your solution has such components, `intlinprog` warns you. If you receive this warning, check the solution to see whether supposedly integer-valued components of the solution are close to integers.

Large Coefficients Disallowed

`intlinprog` does not allow components of the problem, such as coefficients in `f`, `A`, or `ub`, to exceed `1e15` in absolute value. If you try to run `intlinprog` with such a problem, `intlinprog` issues an error.

If you get this error, sometimes you can scale the problem to have smaller coefficients:

- For coefficients in `f` that are too large, try multiplying `f` by a small positive scaling factor.
- For constraint coefficients that are too large, try multiplying all bounds and constraint matrices by the same small positive scaling factor.

References

[1] Williams, H. Paul. *Model Building in Mathematical Programming.* Wiley, 2013.