Visualizing Constraints in Multi-Objective Optimization

Priyanshu Tiwari

2025-03-14

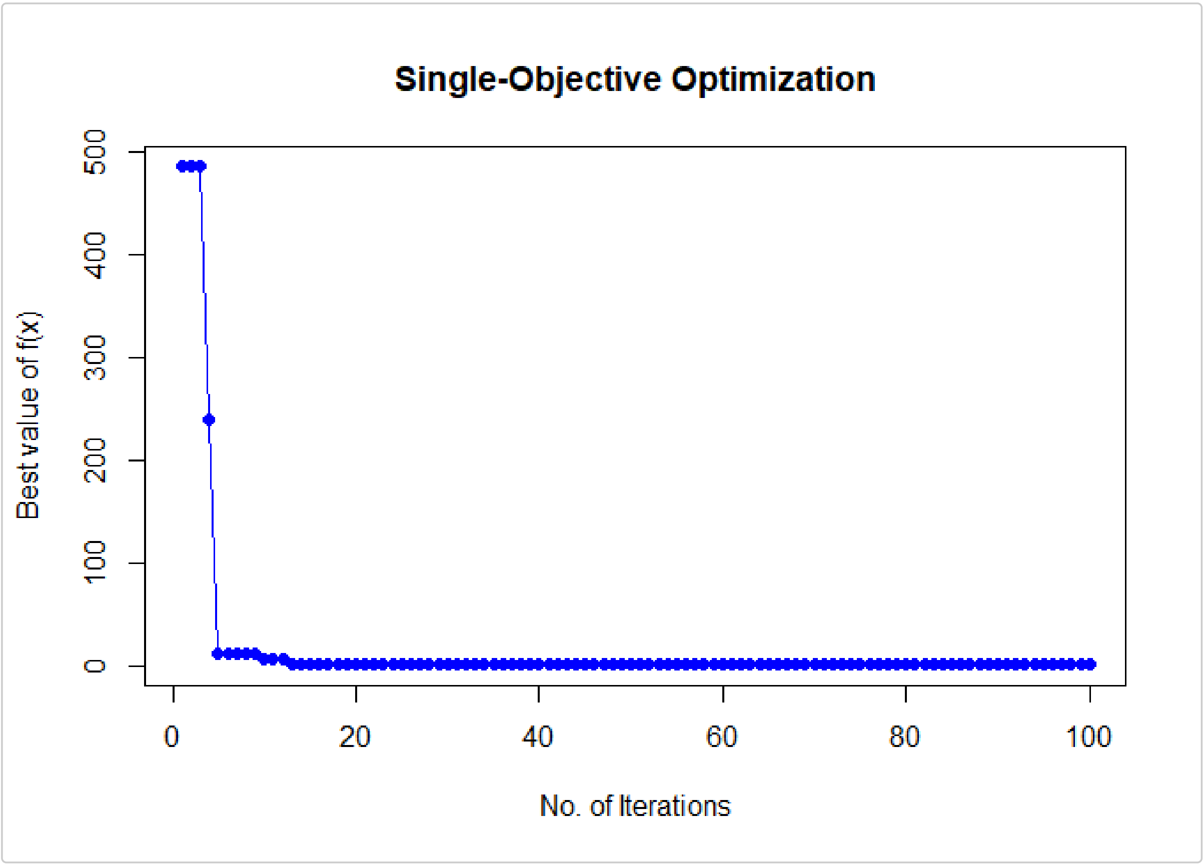
Introduction

In multi-objective optimization, constraints often play a crucial role in determining the set of feasible solutions. Understanding how these constraints shape the solution space is essential for decision-makers. This vignette demonstrates the new constraint visualization functionality in the Jaya package, which helps users gain insights into the relationship between constraints and Pareto-optimal solutions.

Setup

First, let’s load the necessary packages and define a simple multi-objective optimization problem with constraints.

```
library(Jaya)
#> Warning: package 'Jaya' was built under R version 4.4.3

# To source all R files in a directory
r_files <- list.files("D:/GSoC-25/Jaya/jayaEnhancement/R", pattern = "\\.R$", full.names = TRUE)
sapply(r_files, source)
#> Warning: package 'Jaya' is in use and will not be installed
#> Jaya Algorithm
#> Population Size      = 30
#> Number of iterations = 100
#> Number of variables  = 3
#>
#> Objective: minimize
#> Objective Function:
#> [[1]]
#> function (x)
#> {
#>     sum(100 * (x[-length(x)]^2 - x[-1])^2 + (x[-length(x)] -
#>         1)^2)
#> }
#> <bytecode: 0x000001d5905b6f58>
#>
#>
#> Limits:
#> x1 = [-5, 5]
#> x2 = [-5, 5]
#> x3 = [-5, 5]
#>
#> Best Result:
#>       Best.x1   Best.x2   Best.x3 Best.f.x.
#> Best 0.741428 0.5843539 0.3361386 0.3624448
```



```
#>         D:/GSoC-25/Jaya/jayaEnhancement/R/constraint_visualization.R
#> value   ?
#> visible FALSE
#>         D:/GSoC-25/Jaya/jayaEnhancement/R/easy.R
#> value   NULL
#> visible FALSE
#>         D:/GSoC-25/Jaya/jayaEnhancement/R/hello.R
#> value   ?
#> visible FALSE
```

Problem Definition

We'll work with a simple two-objective optimization problem with two variables. The objectives are:

1. Minimize the sum of squares: $f_1(x) = \sum_{i=1}^{n} x_i^2$
2. Minimize the sum of squares from point (2, 2): $f_2(x) = \sum_{i=1}^{n}(x_i - 2)^2$

We'll add two constraints:

1. The sum of variables must be less than or equal to 3: $\sum_{i=1}^{n} x_i \leq 3$
2. The sum of variables must be greater than or equal to 1: $\sum_{i=1}^{n} x_i \geq 1$

These constraints create a bounded region in the decision space, which will impact the available solutions in the objective space.

```
# Define objectives
objective1 <- function(x) sum(x^2)
objective2 <- function(x) sum((x - 2)^2)

# Define constraints (must return <= 0 for feasibility)
constraint1 <- function(x) sum(x) - 3  # Sum of variables must be <= 3
constraint2 <- function(x) -sum(x) + 1 # Sum of variables must be >= 1

# Set variable bounds
```

```r
lower_bounds <- c(-5, -5)
upper_bounds <- c(5, 5)
```

Running the Enhanced Multi-Objective Optimization

Now, we’ll run the optimization using the enhanced version of the Jaya algorithm that tracks constraint information throughout the optimization process.

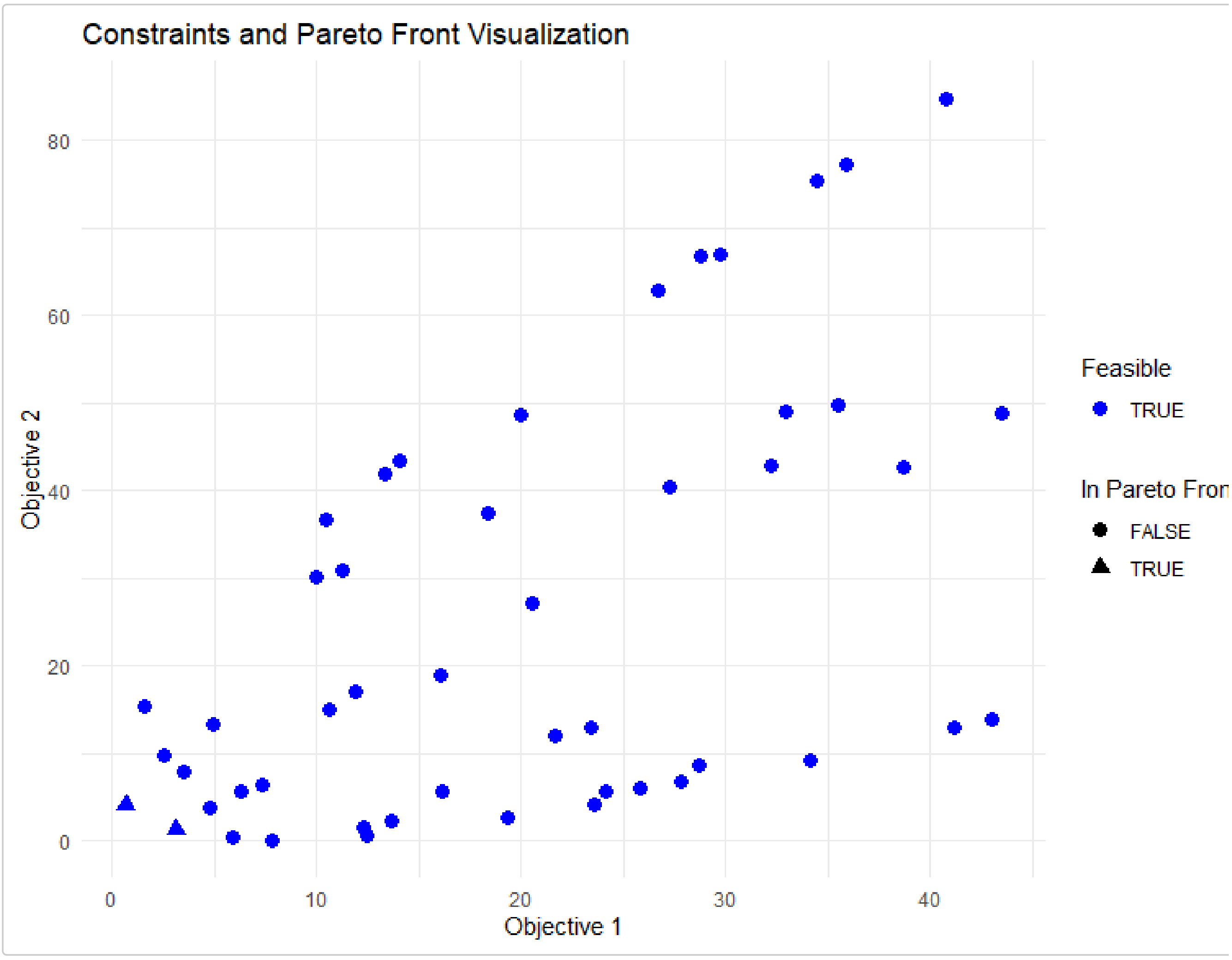
```r
set.seed(123) # For reproducibility
result <- jaya_multi_enhanced(
  objectives = list(objective1, objective2),
  lower = lower_bounds,
  upper = upper_bounds,
  popSize = 50,
  maxiter = 100,
  n_var = 2,
  constraints = list(constraint1, constraint2)
)

# Examine the structure of the result
str(result, max.level = 1)
#> List of 2
#>  $ Pareto_Front:'data.frame':    25 obs. of  4 variables:
#>  $ Solutions   :'data.frame':    50 obs. of  4 variables:
#>  - attr(*, "popSize")= num 50
#>  - attr(*, "maxiter")= num 100
#>  - attr(*, "n_var")= num 2
#>  - attr(*, "lower")= num [1:2] -5 -5
#>  - attr(*, "upper")= num [1:2] 5 5
#>  - attr(*, "objectives")=List of 2
#>  - attr(*, "constraints")=List of 2
#>  - attr(*, "constraint_info")=List of 3
#>  - attr(*, "class")= chr "jaya_multi"
```

The result object contains the same components as the original `jaya_multi` function, but with additional constraint information stored as attributes. This information is used by the visualization functions to show how constraints affect the solution space.

Basic Constraint Visualization

Let’s visualize the Pareto front and the impact of constraints using the new `plot_jaya_constraints` function.

```r
# Generate the constraint visualization
p <- plot_jaya_constraints(result)
print(p)
```



In this visualization:

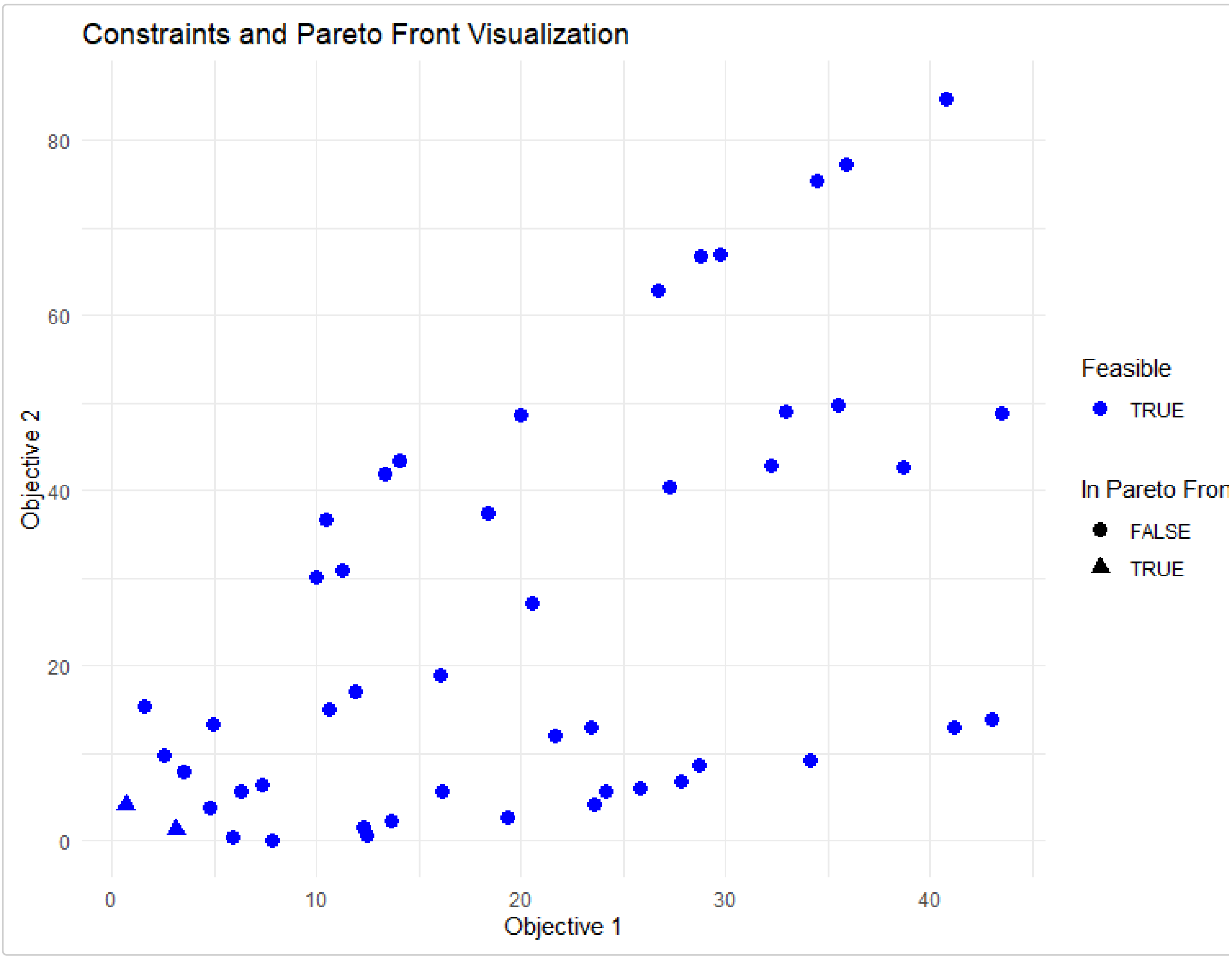
- Blue points represent feasible solutions that are part of the Pareto front
- Red points represent infeasible solutions that violate at least one constraint
- The green dashed lines (if present) represent the approximate boundaries of the feasible region defined by the constraints

You can see how the constraints limit the solutions in the objective space. Without constraints, the Pareto front would extend further, but the constraints create a boundary that limits the achievable objective values.

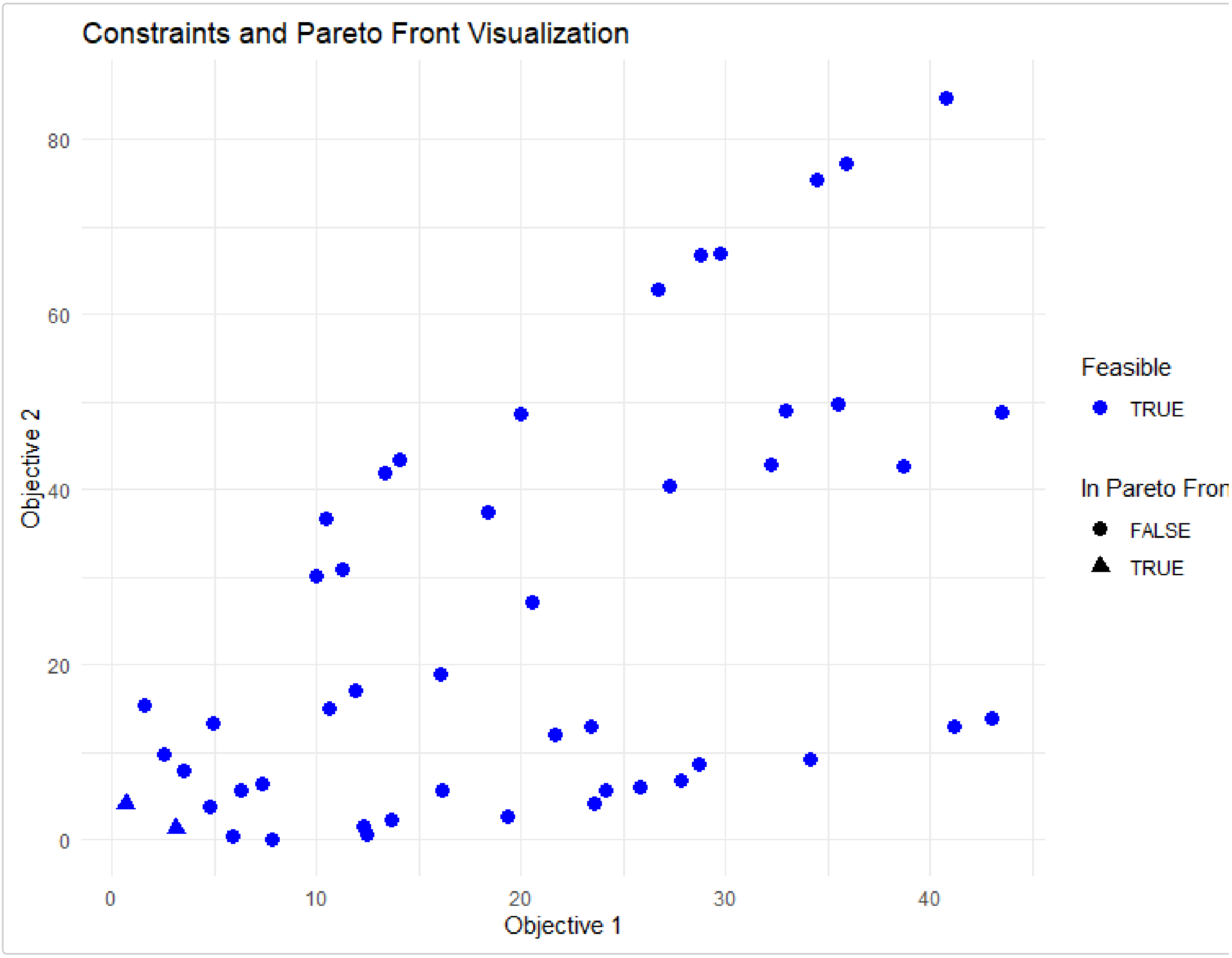
Focusing on Specific Constraints

We can also focus on specific constraints to understand their individual impact on the solution space.

```
# Visualize only the first constraint
p1 <- plot_jaya_constraints(result, constraints_to_show = 1)
print(p1)
```



```r
# Visualize only the second constraint
p2 <- plot_jaya_constraints(result, constraints_to_show = 2)
print(p2)
```



By examining each constraint separately, we can identify which constraints are most restrictive and better understand how they shape the Pareto front.

Interactive Visualization

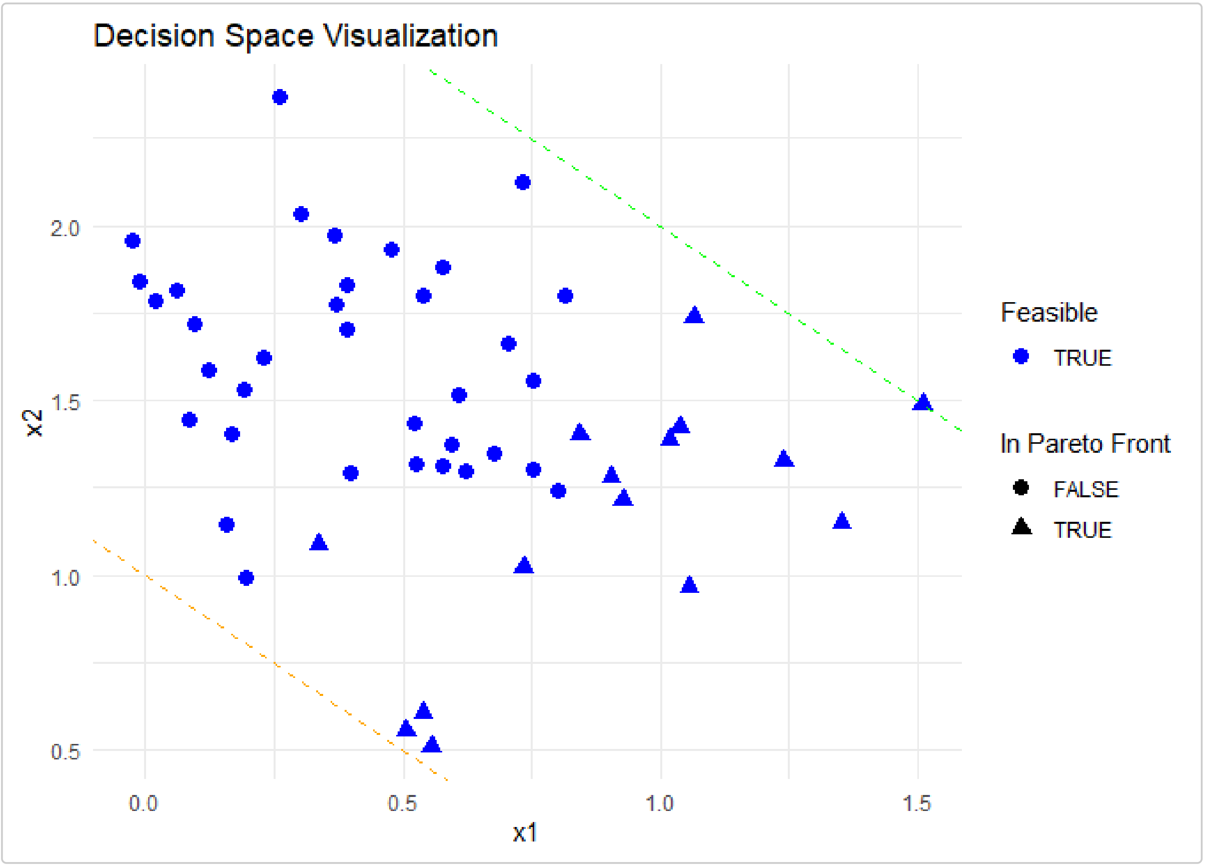
The `plot_jaya_constraints` function also supports interactive visualization using the plotly package, which allows for zooming, panning, and hovering to see exact values.

```r
# Create an interactive plot
interactive_plot <- plot_jaya_constraints(result, interactive = TRUE)
interactive_plot
```

Understanding the Decision Space

While the previous visualizations show the objective space, it's often useful to understand how constraints affect the decision variables directly. We can extract this information from the solution set:

```r
# Create a data frame with decision variables and feasibility information
decision_data <- data.frame(
  x1 = result$Solutions$x1,
  x2 = result$Solutions$x2,
  Feasible = attr(result, "constraint_info")$feasible,
  InPareto = FALSE
)

# Mark points in the Pareto front
for (i in 1:nrow(result$Pareto_Front)) {
  for (j in 1:nrow(result$Solutions)) {
    if (all(result$Solutions[j, c("x1", "x2")] == result$Pareto_Front[i, c("x1", "x2")])) {
      decision_data$InPareto[j] <- TRUE
      break
    }
  }
}

# Plot the decision space
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(decision_data, aes(x = x1, y = x2, color = Feasible, shape = InPareto)) +
    geom_point(size = 3) +
    scale_color_manual(values = c("FALSE" = "red", "TRUE" = "blue"), name = "Feasible") +
    scale_shape_manual(values = c("FALSE" = 16, "TRUE" = 17), name = "In Pareto Front") +
    theme_minimal() +
    labs(
      title = "Decision Space Visualization",
      x = "x1",
      y = "x2"
    ) +
    # Add constraint boundaries
    geom_abline(intercept = 3, slope = -1, color = "green", linetype = "dashed") +  # sum(x) = 3
    geom_abline(intercept = 1, slope = -1, color = "orange", linetype = "dashed")   # sum(x) = 1
}
```

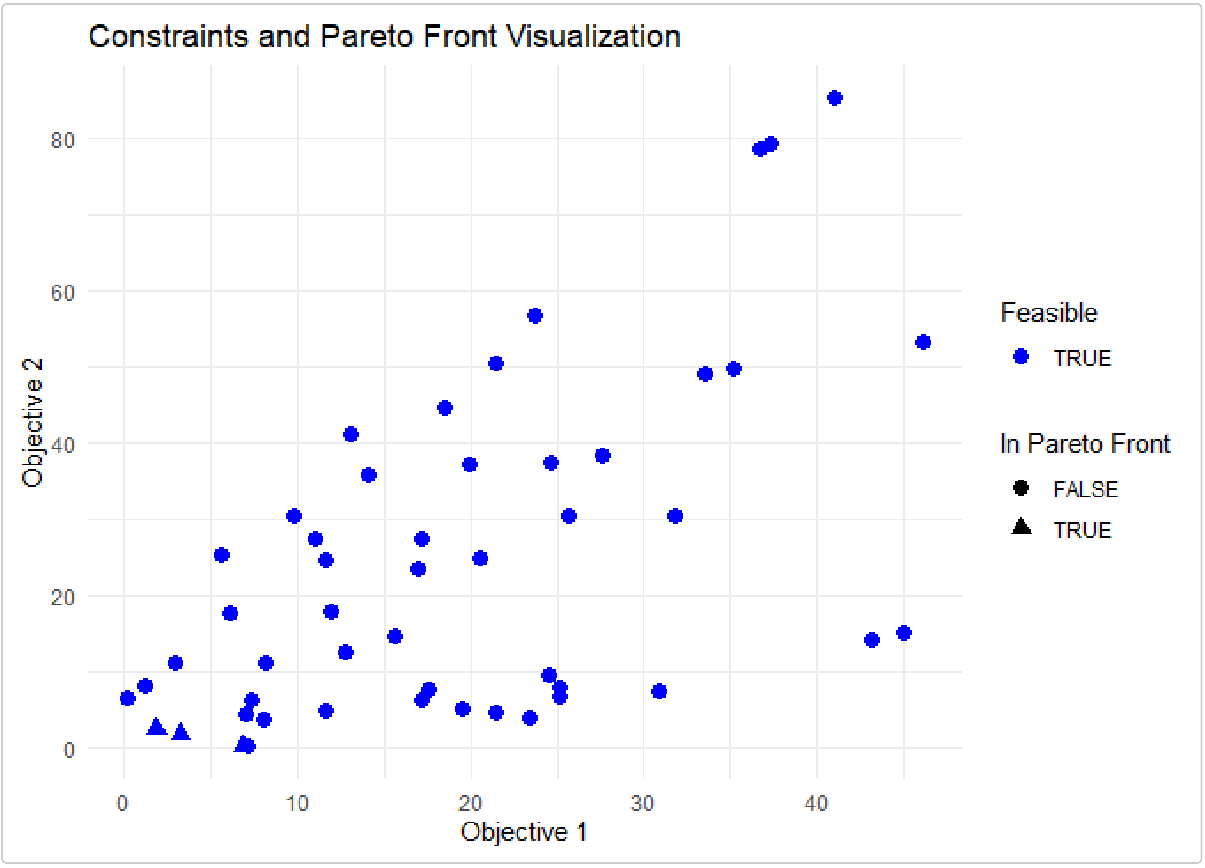


In this decision space visualization, we can see: - The actual values of the decision variables - Which solutions are feasible (blue) and which are infeasible (red) - Which solutions are part of the Pareto front (triangles) - The boundaries created by our constraints (dashed lines)

Sensitivity Analysis

One of the powerful applications of constraint visualization is sensitivity analysis. By relaxing or tightening constraints, we can see how the Pareto front changes. Let's modify our first constraint and compare the results:

```r
# Define a relaxed version of constraint1
relaxed_constraint1 <- function(x) sum(x) - 5  # Sum of variables must be <= 5

# Run optimization with the relaxed constraint
result_relaxed <- jaya_multi_enhanced(
  objectives = list(objective1, objective2),
  lower = lower_bounds,
  upper = upper_bounds,
  popSize = 50,
  maxiter = 100,
  n_var = 2,
  constraints = list(relaxed_constraint1, constraint2)
)

# Visualize the relaxed constraint results
p_relaxed <- plot_jaya_constraints(result_relaxed)
print(p_relaxed)
```



Compare this with our original result to see how relaxing the constraint expands the feasible region and potentially improves the Pareto front.

Conclusion

The constraint visualization functionality in the Jaya package provides valuable insights into how constraints affect multi-objective optimization problems. By visualizing the relationship between constraints and the Pareto front, users can:

1. Understand which constraints are most restrictive
2. Identify potential areas for constraint relaxation
3. Gain intuition about the trade-offs between objectives under constraints
4. Make more informed decisions based on the available solution space

This tool enhances the interpretability of optimization results and can guide decision-makers toward solutions that best balance their objectives while respecting constraints.