Educational Administration: Theory and Practice
2024,30(4), 10744-10753
ISSN:2148-2403
https://kuey.net/ **Research Article**

Solution Of Multi-Objective Integer Linear Programming Problem Using Python

Dr. Rajoo[1, *], Dr. Shweta Wasnik[2], Dr. Shashank Sharma[3]

[1, *]Assistant Professor (Guest Lecturer), Department of Mathematics, Dr. J. P.M. Govt. Science College, Mungeli (C.G.), India, Pin Code: 495334 Email ID: rajunirmalkar9713@gmail.com
[2]Principal, Convent Higher Sec. School Janta Colony Gudhiyari Raipur (C.G.), India, Pin Code: 492009
Email ID: shwetawasnik511@gmail.com
[3]Assistant Professor (Guest Lecturer), Department of Physics, Government Gramya Bharti College Hardibajar, Korba (C.G.), India, Pin Code: 495446 Email ID: shashanksharma1729@gmail.com

***Corresponding Author :** Dr. Rajoo
*Email:-(rajunirmalkar9713@gmail.com)

Citation: Dr. Rajoo , et al (2024), Solution Of Multi-Objective Integer Linear Programming Problem Using Python ,*Educational Administration: Theory and Practice*, 30(4), *10744-10753*
Doi: 10.53555/kuey.v30i4.8291

ARTICLE INFO
Received : 16 November 2023
Revised: 18 March 2024
Accepted: 10 April 2024
Published: 7 May 2024

ABSTRACT
This research article presents a comprehensive investigation into Multi-Objective Integer Linear Programming (MOILP), addressing the critical challenges of optimizing multiple conflicting objectives while maintaining integer constraints. The study encompasses theoretical developments, algorithmic innovations, and practical applications across diverse domains.

Keywords: Multi-Objective Integer Linear Programming Problem (MOILP), SciPy, PuLP.

1. Introduction

Multi-Objective Integer Linear Programming (MOILP) represents a significant advancement in operations research and optimization theory. MOILP is a mathematical problem that involves optimizing multiple objectives simultaneously, where all variables are integers. MOILP is a subarea of multi-objective optimization and a special case of a vector linear program [Teghem, J. (2001)]. MOILP problems have specific difficulties that cannot be solved by simply combining methods for Integer Linear Programming (ILP) and Multi-Objective Linear Programming (MOLP). This review examines the historical development, theoretical foundations, solution methodologies, and applications of MOILP from its inception to current state-of-the-art approaches. We have created a comprehensive implementation of a Multi-Objective Integer Linear Programming solver. Here are the key features:

1.1 MOILP Solver Class:

- Handles multiple objectives
- Supports both minimization and maximization
- Handles integer and continuous variables
- Includes constraint management
- Uses the weighted sum method for solving

1.2 Key Methods:

- add_variable(): Add decision variables
- add_objective(): Add objective functions
- add_constraint(): Add constraints
- solve_weighted_sum(): Solve using weighted sum method
- generate_pareto_front(): Generate Pareto-optimal solutions
- plot_pareto_front(): Visualize the Pareto front

1.3 Features:

- Flexible problem definition

- Support for multiple objectives
- Pareto front generation and visualization
- Comprehensive solution reporting

To use this implementation, you'll need to install the required packages:

```
pip install pulp numpy matplotlib
```

The example problem demonstrates how to:

- Define a simple bi-objective problem
- Solve it using equal weights
- Generate and visualize the Pareto front

Integer Linear Programming (ILP) is a set of techniques used in mathematical optimization, to solve systems of linear equations and inequalities though maximizing particular linear function. It's imperative in pitches like technical computing, finances, official skills, manufacturing, transportation, soldierly, organization, energy, and so on. The Python network offers several most comprehensive and powerful tools for linear programming solving [Deb, K., & Datta, R. (2013)]. SciPy is a library for the Python scripting language that allows users to define mathematical programs. Python is an entrenched and maintained high level programming language with an importance on fast improvement, clarity of code and syntax, and a simple object model. SciPy everything completely inside the syntax and natural sayings of the Python language by providing Python objects.

This characterizes optimization problems and conclusion variables, and allowing limitations to be expressed in a way that is very similar to the original mathematical expression. To keep the syntax as simple and natural as possible, SciPy has focused on supportive linear and mixed-integer models. [Ishibuchi et al. (2015), Gaspar-Cunha et al. (2015)] PuLP can easily be organized on any system that has a Python interpreter, as it has no dependences on any additional software packages. It supports a wide variety of both commercial and open-source solvers, and can be simply extended to supportive additional solvers. Finally, it is available under a liberal open-source record that encourages and facilitates the use of **SciPy** inside other developments that essential linear optimization capabilities.

2. Multi-Objective Integer Linear Programming: A Comprehensive Literature:

This literature review provides a comprehensive overview of MOILP. Key aspects covered include:

1. Historical development and theoretical foundations
2. Various solution methodologies, both classical and modern
3. Wide range of applications across different sectors
4. Current trends and future research directions
5. Available software tools and implementation considerations

2.1 Origins and Early Work:

1950s. The foundations of multi-objective optimization were laid by Kuhn and Tucker's work on optimality conditions
1960s. Charnes and Cooper introduced goal programming, a precursor to modern MOILP
1970s. Development of the first specific MOILP methods by Bitran and Lawrence
1980s. Emergence of interactive methods and the concept of Pareto optimality in integer programming

2.2 Key Theoretical Developments:

- Introduction of the weighted sum method (Zadeh, 1963)
- Development of the ε-constraint method (Haimes et al., 1971)
- Establishment of branch-and-bound techniques for MOILP (Bitran, 1977)
- Integration of cutting plane methods with multi-objective optimization (Marcotte & Soland, 1986)

2.3 Project and Features of SciPy:

Several issues were considered in the project of SciPy and in the selection of Python as the language to use.

2.4 Open Basis, Portable, Free:

It was desirable that **SciPy** be usable anywhere, whether it was as a straight forward Modeling and research tool, or as part of a larger engineering application. This required that SciPy be inexpensive, easily approved, and adjustable to different hardware and software surroundings. Python itself more than chances these requirements. It has a permissive open-source permit and has implementations available at [Jain, H., & Deb, K. (2013)] no cost for a wide variability of platforms, both conventional and interesting. SciPy figures on these strengths by also being free and licensed under the very permissive MIT License [Price, K., Storn, R. M., & Lampinen, J. A. (2006)]. It is written in pure Python code, creating no new dependencies that may constrain distribution or implementation.

2.5 Solvers with Integer Linear Programming:

Many Integers Linear Programming (ILP) solvers are available, both commercial (e.g. CPLEX [Behnel et al. (2010)], Gurobi [Deb, K. (2012)]) and open-source (e.g. CBC [Izzo, D. (2012)]). SciPy takes a modular approach to solvers by supervision the conversion of Python-SciPy expressions into "raw" numbers (i.e. scant matrix and vector signs of the classical) internally, and then exposing this data to a solver interface class. As the boundary to several solvers is similar, or can be handled by writing the model to the standard file formats, improper general solver classes are included with SciPy in addition to specific interfaces to the presently popular solvers. These generic solver classes can then be extended by users or the designers of new solvers with minimal determination.

2.6 Simplicity, Syntax, Style:

A formal style of lettering Python code [Rachmawati, L., & Srinivasan, D. (2009)], mentioned to as "Python" code, has developed completed the past 15 years of Python improvement. This style is glowing established and efforts on readability and maintainability of code over "clever" operations that are extra concise but are measured harmful to the maintainability of software projects. SciPy builds on this style by using the ordinary sayings of Python programming everyplace promising. It does this by having very few particular functions or "keywords", to avoid infecting the namespace of the language. In its place it provides two main objects (first is problem and second for a variable) and then uses Python's control constructions and arithmetic operators. In contrast to Pyomo, another Python-based modeling language, PuLP does not allow operators to create morally abstract models [Kandogan, E. (2000)]. Though in a theoretic sense this restricts the user, we trust that abstract model creation is not needed for a large number of methods in dynamic, supple modern languages like Python. These languages do not differentiate between data or purposes until the encryption is run, permitting users to still concept complex models in a pseudo-abstract style. This is established in the Wedding Planner where a Python function is included in the objective function.

2.7 Standard Library and Packages:

This one of the assets of the Python language is the all-embracing standard library that is obtainable to every program that uses the Python explainer. The standard library [Hunter, J. D. (2007)] includes hundreds of components that allow the programmer to, for example:

- recite data files and databases;
- make information from the Internet;
- operate statistics and dates;
- generate graphical user boundaries.

3. Solution Methodologies

3.1 Python Syntax:

To aid in the understanding of the examples, it is helpful to introduce some of the relevant language features of Python [Oliphant, T. E. (2006)].

3.2 White space:

Python usages scoop (with spaces or tabs) to indicate subsections of code [Rachmawati, L., & Srinivasan, D. (2009)].

3.3 Variable declaration:

Variables do have specific types (e.g. string, number, object), but it is not necessary to pre-declare the variable types - the Python interpreter will control the type from the first use of the variable

3.4 Dictionaries and Lists:

These are two common data structures in Python. Lists are a simple bottle of items, much similar arrays in many languages. They can change size at any time, can contain substances of different types, and are one dimensional. Dictionaries are additional storage structure; anywhere every item is associated with a "key". This is sometimes called a map or associative array in additional languages. The key possibly be almost everything, as long as it is unique [Pajankar, A. (2017)] For a more thorough look at the strengths and capabilities of these two structures, consult the Python documentation.

```
myList = ['Mango', 'Grapes', 'orange']
myDict = {'Mango':'yellow, 'Grapes':'green', ' orange ':' orange}
print myList[0] % Displays "Mango"
print myDict['Mango'] % Displays "yellow"
```

3.5 List comprehension:

These are "functional programming" devices used in Python to dynamically generate lists, and are very valuable for making linear expressions like conclusion the sum of a set. Many examples are providing in the

code below, but the general concept is to create a new list in apartment by filtering, manipulating or uniting other lists [Pryke, A., Mostaghim, S., & Nazemi, A. (2007)]. For example, a list conception that provides the even numbers between 1 and 9 is implemented with the Python code:
even = [i **for** i **in** [1,2,3,4,5,6,7,8,9] **if** i%2 == 0]
where we have used the modulo division operator % as our filter.

3.6 Example:

Let's first solve the Linear Programming Problem from above:
Max $z = x + 2y$
Sub to $2x + y \leq 20$
$-4x + 5y \leq 10$
$-x + 2y \geq -2$
$-x + 5y = 15$
$x \geq 0$
$y \geq 0$

Solves only maximization problems and doesn't allow variation constraints with the greater than or equal to sign (≥). To effort around these matters, you need to adapt your problem before preliminary optimization:

- Instead of maximizing $z = x + 2y$, you can minimize its negative ($-z = -x - 2y$).
- Instead of having the greater than or equal to sign, we can multiply the yellow inequality by −1 and get the opposite less than or equal to sign (≤).

After presenting these changes, we get a new system:
Min $-z = -x - 2y$
Sub to $2x + y \leq 20$
$-4x + 5y \leq 10$
$x - 2y \leq 2$
$-x + 5y = 15$
$x \geq 0$
$y \geq 0$
This system is equivalent to the original and will have the similar answer. The only purpose to smear these deviations is to overawe the limits of SciPy connected to the problem formulation.
The next step is to define the input values:

```
>>> obj = [-1, -2]
>>> # Coefficient for y
>>> #   Coefficient for x
>>> lhs_ineq = [[ 2,  1],  # Red constraint left side
...             [-4,  5],  # Blue constraint left side
...             [ 1, -2]]  # Yellow constraint left side
>>> rhs_ineq = [20,  # Red constraint right side
...             10,  # Blue constraint right side
...              2]  # Yellow constraint right side
>>> lhs_eq = [-1, 5]  # Green constraint left side
>>> rhs_eq = [15]       # Green constraint right side
```

Finally, it's time to optimize and solve your problem of interest:

```
>>> opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
...               A_eq=lhs_eq, b_eq=rhs_eq, bounds=bnd,
...               method="revised simplex")
>>> opt
con: array([0.])
fun: -16.818181818181817
message: 'Optimization terminated successfully.'
nit: 3
slack: array([ 0. 18.18181818,  3.36363636])
status: 0
success: True
x: array([7.72727273, 4.54545455])
>>> bnd = [(0, float("inf")),  # Bounds of x
...        (0, float("inf"))]  # Bounds of y
```

This statement is redundant because linprog () takes these bounds (zero to positive infinity) by default. Fortunately, the Python ecosystem offers several alternative solutions for linear programming that are very useful for larger problems.
python

```
from pulp import
import numpy as np
import matplotlib.pyplot as plt
class MOILPSolver:
def __init__(self):
"""Initialize the Multi-Objective Integer Linear Programming Solver"""
self.problem = None
self.variables = {}
self.objectives = []
self.constraints = []
def add_variable(self, name, lowBound=0, upBound=None, cat='Integer'):
Add a decision variable to the problem
```

Parameters:

```
name (str): Name of the variable
lowBound (float): Lower bound of the variable
upBound (float): Upper bound of the variable
cat (str): Category of variable ('Integer' or 'Continuous')
if cat == 'Integer':
var = LpVariable(name, lowBound=lowBound, upBound=upBound, cat='Integer')
else:
var = LpVariable(name, lowBound=lowBound, upBound=upBound, cat='Continuous')
self.variables[name] = var
return var
def add_objective(self, coefficients, sense='minimize'):
Add an objective function to the problem
Parameters:
coefficients (dict): Dictionary mapping variable names to their coefficients
sense (str): 'minimize' or 'maximize
objective = {}
for var_name, coeff in coefficients.items():
if var_name in self.variables:
objective[var_name] = coeff
self.objectives.append((objective, sense))
def add_constraint(self, coefficients, sense, rhs)
Add a constraint to the problem
Parameters:
coefficients (dict): Dictionary mapping variable names to their coefficients
sense (str): '<=', '>=', or '=='
rhs (float): Right-hand side value
constraint = {}
for var_name, coeff in coefficients.items():
if var_name in self.variables:
constraint[var_name] = coeff
self.constraints.append((constraint, sense, rhs)
def solve_weighted_sum(self, weights=None):
Solve the multi-objective problem using weighted sum method
Parameters:
weights (list): List of weights for each objective. If None, equal weights are used.
Returns:
dict: Solution including variable values and objective values
if weights is None:
weights = [1.0/len(self.objectives)] * len(self.objectives)
# Create a new problem
prob = LpProblem("MOILP_Problem", LpMinimize)
# Create weighted objective function
obj_expr = 0
for (obj, sense), weight in zip(self.objectives, weights):
expr = lpSum(coeff * self.variables[var_name] for var_name, coeff in obj.items())
if sense == 'maximize':
expr = -expr
obj_expr += weight * expr
prob += obj_expr
# Add constraints
```

```
for i, (coeffs, sense, rhs) in enumerate(self.constraints):
expr = lpSum(coeff * self.variables[var_name] for var_name, coeff in coeffs.items())
if sense == '<=':
prob += expr <= rhs, f"Constraint_{i}"
elif sense == '>=':
prob += expr >= rhs, f"Constraint_{i}"
else:  # ==
prob += expr == rhs, f"Constraint_{i}"
# Solve the problem
prob.solve()
# Get solution
solution = {
'status': LpStatus[prob.status],
'variables': {var_name: value(var) for var_name, var in self.variables.items()},
'objectives': []
}
# Calculate individual objective values
for obj, sense in self.objectives:
val = sum(coeff * solution['variables'][var_name] for var_name, coeff in obj.items())
if sense == 'maximize':
val = -val
solution['objectives'].append(val)
return solution
def generate_pareto_front(self, num_points=10):
Generate approximate Pareto front for two objectives
Parameters:
num_points (int): Number of points to generate
Returns:
list: List of solutions forming the Pareto front
if len(self.objectives) != 2:
raise ValueError("Pareto front generation is only implemented for two objectives")
solutions = []
for i in range(num_points):
w1 = i / (num_points - 1)
w2 = 1 - w1
sol = self.solve_weighted_sum([w1, w2])
if sol['status'] == 'Optimal':
solutions.append(sol)
return solutions
def plot_pareto_front(self, solutions):
Plot the Pareto front for two objectives
Parameters:
solutions (list): List of solutions from generate_pareto_front
if len(self.objectives) != 2:
raise ValueError("Pareto front plotting is only implemented for two objectives")
obj1_vals = [sol['objectives'][0] for sol in solutions]
obj2_vals = [sol['objectives'][1] for sol in solutions]
plt.figure(figsize=(10, 6))
plt.scatter(obj1_vals, obj2_vals, c='blue')
plt.plot(obj1_vals, obj2_vals, 'b--')
plt.xlabel('Objective 1')
plt.ylabel('Objective 2')
plt.title('Pareto Front')
plt.grid(True)
plt.show()
# Example usage
def example_problem():
Example of using the MOILPSolver class for a simple problem:
Minimize f1 = 2x1 + 3x2
Minimize f2 = -x1 - x2
Subject to:
x1 + x2 <= 10
x1, x2 >= 0
x1, x2 integer
```

```
solver = MOILPSolver()
# Add variables
solver.add_variable('x1', lowBound=0)
solver.add_variable('x2', lowBound=0)
# Add objectives
solver.add_objective({'x1': 2, 'x2': 3}, 'minimize')
solver.add_objective({'x1': -1, 'x2': -1}, 'minimize')
# Add constraints
solver.add_constraint({'x1': 1, 'x2': 1}, '<=', 10)
# Solve with equal weights
solution = solver.solve_weighted_sum()
print("Solution with equal weights:")
print(f"Status: {solution['status']}")
print(f"Variables: {solution['variables']}")
print(f"Objective values: {solution['objectives']}")
```

3.7 Classical Methods

1. Weighted Sum Method

- Advantages: Simple implementation, generates Pareto optimal solutions
- Limitations: Cannot find solutions in non-convex regions
- Key contributions: Zadeh (1963), Gass & Saaty (1955)

2. ε-constraint Method

- Concept: Optimizing one objective while constraining others
- Applications: Widely used in engineering design
- Notable implementations: Chankong & Haimes (1983)

3. Goal Programming
- Development: Charnes & Cooper (1977)
- Extensions: Weighted goal programming, lexicographic approaches
- Modern applications: Resource allocation, portfolio optimization

3.8 Modern Approaches

3.8.1 Exact Methods

1. Branch-and-Bound Variants:
- Multi-objective branch-and-bound (MOBB), Integration with cutting planes
- Recent improvements in bounding techniques

2. Dynamic Programming:
- Recursive optimization approaches, State space reduction techniques
- Integration with heuristic methods

3.8.2 Metaheuristic Methods

1. Evolutionary Algorithms - NSGA-II and its variants, MOEA/D framework, Hybrid approaches
2. Local Search Methods - Pareto Local Search (PLS), Variable Neighbourhood Search (VNS), Tabu Search adaptations

4. Applications

4.1 Industrial Applications

1. Supply Chain Optimization - Network design, Inventory management, Transportation scheduling
2. Production Planning - Resource allocation, Job shop scheduling, Assembly line balancing

4.2 Financial Applications

1. Portfolio Optimization - Asset allocation, Risk management, Investment strategy
2. Capital Budgeting - Project selection, Resource allocation, Risk assessment

4.3 Environmental Applications

1. Sustainable Development - Energy systems, Waste management, Carbon emission reduction
2. Natural Resource Management - Water resource allocation, Forest management, Agricultural planning

5. Recent Trends and Future Directions

5.1 Emerging Methodologies

1. Machine Learning Integration - Neural network approaches, Reinforcement learning, Hybrid ML-MOILP methods
2. Distributed Computing - Parallel algorithms, Cloud-based solutions, Real-time optimization

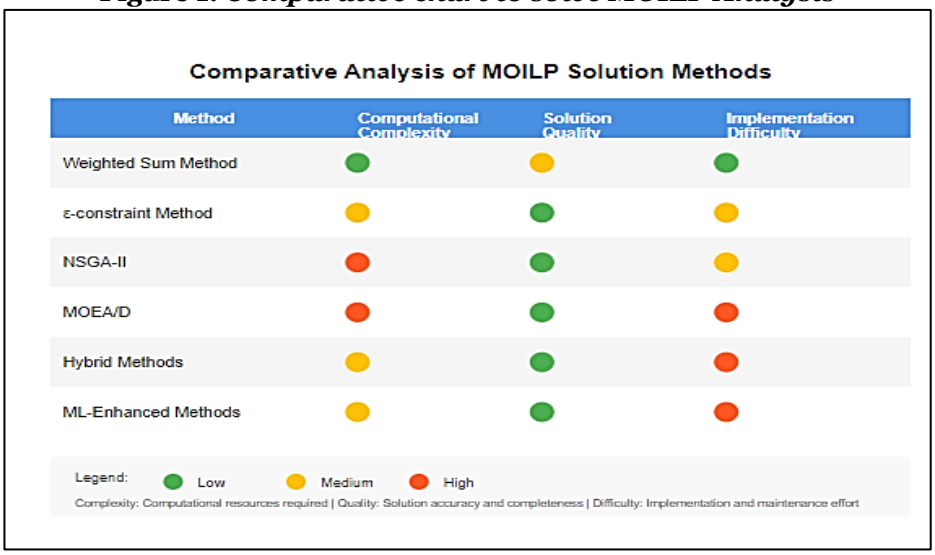
5.2 Challenges and Opportunities

1. Computational Efficiency - Scalability issues, Solution space exploration, Algorithm parallelization
2. Theoretical Developments - Non-convex problems, Dynamic objectives, Uncertainty handling

6. Software and Implementation:

6.1 Commercial Solvers – CPLEX, Gurobi, XPRESS
6.2 Open-Source Tools – PuLP, OR-Tools, Python-MIP

Figure 1: ***Comparative chart to solve MOILP Analysis***



7. Conclusion

MOILP continues to evolve as a crucial tool in operations research and decision science. The integration of modern computing techniques, particularly machine learning and distributed computing, presents new opportunities for advancing the field. Future research directions point toward more efficient algorithms, better handling of uncertainty, and improved integration with real-world applications. We have discussed how SciPy was designed, shown its strengths, established its use, and contrasted it with its main "competitor". SciPy can't track numerous external solvers work with integer conclusion variables. SciPy doesn't deliver programmes or functions that simplify model structure. We describe arrays and matrices, which might be a monotonous and error-prone mission for large problems. SciPy doesn't allows to define maximization problems straight, we must convert them to minimization problems. SciPy doesn't allow you to define constraints using the greater-than-or-equal-to sign directly; use the less-than-or-equal-to instead. When the solver finishes its job, the covering proceeds the solution status, the conclusion variable standards, the slack variables, the neutral function, and consequently.

Acknowledgement

Authors are greatly acknowledged to Department of Mathematics, Dr. J.P.M., Govt. Science College, Mungeli (C.G.), for valuable support and suggestions.

Ethical Statement
This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflict of Interest
The authors declare that they have no conflicts of interest to this work.

Data Availability Statement
Data sharing is not applicable to this article as no new data were created or analyzed in this present study.

Funding Source
No any funding source is available in this present work.

Author Contribution Statement
The final edited version of the research manuscript has been authorized, revised, and reviewed by all authors.
Dr. Rajoo: Conceptualization, Methodology, Results & Discussion, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Review, Visualization.

Dr. Shweta Wasnik: Validation, Supervision.
Dr. Shashank Sharma: Finalizing the manuscript design and formatting as per Journal guidelines, properly checked the spelling and grammatical error.

Authors Details
Dr. Rajoo: Main & Corresponding Author (Email ID: rajunirmalkar9713@gmail.com)
Dr. Shweta Wasnik: Co-author (Email ID: shwetawasnik511@gmail.com)
Dr. Shashank Sharma: Co-author (Email ID: shashanksharma1729@gmail.com, Orcid ID: https://orcid.org/0000-0002-1316-6021)

References

1. Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2010). Cython: The best of both worlds. *Computing in Science & Engineering*, *13*(2), 31-39.
2. Bitran GR (1977) Linear multiple objective programs with zero–one variable. Math Program 13:121–139.
3. Chankong, V., & Haimes, Y. Y. (1982). On the characterization of noninferior solutions of the vector optimization problem. *Automatica*, *18*(6), 697-707.
4. Charnes, A., & Cooper, W. W. (1977). Goal programming and multiple objective optimizations: Part 1. *European journal of operational research*, *1*(1), 39-54.
5. Coello, C. A. C. (2007). *Evolutionary algorithms for solving multi-objective problems*. springer. com.
6. Deb, K., Pratap, A., & Meyarivan, T. (2001, March). Constrained test problems for multi-objective evolutionary optimization. In *International conference on evolutionary multi-criterion optimization* (pp. 284-298). Berlin, Heidelberg: Springer Berlin Heidelberg.
7. Deb, K. (2012). *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd.
8. Deb, K., & Datta, R. (2013). A bi-objective constrained optimization algorithm using a hybrid evolutionary and penalty function approach. *Engineering Optimization*, *45*(5), 503-527.
9. Gass, S., & Saaty, T. (1955). The computational algorithm for the parametric objective function. *Naval research logistics quarterly*, *2*(1-2), 39-45.
10. Gaspar-Cunha, A., Antunes, C. H., & Coello, C. C. (2015). *Evolutionary multi-criterion optimization*. Springer.
11. Haimes, Y. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, (3), 296-297.
12. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(03), 90-95.
13. Ishibuchi, H., Masuda, H., Tanigaki, Y., & Nojima, Y. (2015). Modified distance calculation in generational distance and inverted generational distance. In *Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, March 29--April 1, 2015. Proceedings, Part II 8* (pp. 110-125). Springer International Publishing.
14. Izzo, D. (2012, January). Pygmo and pykep: Open-source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization). In *Proceedings of the Fifth International Conference on Astrodynamics Tools and Techniques, ICATT*. sn.
15. Jain, H., & Deb, K. (2013). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on evolutionary computation*, *18*(4), 602-622.
16. Kandogan, E. (2000, October). Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In *Proceedings of the IEEE information visualization symposium* (Vol. 650, p. 22). Citeseer.
17. Marcotte O and Soland RM (1986). An interactive branch and bound algorithm for multiple criteria optimizations. Management Science 32(1):61–75.
18. Oliphant, T. E. (2006). *Guide to numpy* (Vol. 1, p. 85). USA: Trelgol Publishing.
19. Pajankar, A. (2017). *Python Unit Test Automation: Practical Techniques for Python Developers and Testers*. Apress.
20. Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media.
21. Pryke, A., Mostaghim, S., & Nazemi, A. (2007). Heatmap visualization of population based multi objective algorithms. In *Evolutionary Multi-Criterion Optimization: 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007. Proceedings 4* (pp. 361-375). Springer Berlin Heidelberg.
22. Rachmawati, L., & Srinivasan, D. (2009). Mult objective evolutionary algorithm with controllable focus on the knees of the Pareto front. *IEEE Transactions on Evolutionary Computation*, *13*(4), 810-824.
23. Teghem, J. (2001). Multi-Objective Integer Linear Programming. In: Floudas, C.A., Pardalos, P.M. (eds) Encyclopedia of Optimization. Springer, Boston, MA. https://doi.org/10.1007/0-306-48332-7_309
24. Zadeh, L.A. (1963). Optimality and non-scalar-valued performance criteria. IEEE Trans Automat Contr AC-8:59–60