IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 28, Issue 1, Ser. 2 (Jan. – Feb. 2026), PP 01-15
www.iosrjournals.org

Optimization Of Graph And Operational Research Problem Solving Using The Optsolver 1.0 Tool

Mutombo Shanga Eddy, Mwangu Shanga Gabriel, Loso Kapita Percy, Kanku Muamba Fidèle
Higher Institute Of Applied Techniques Of Gombe Matadi, Mbanza-Ngungu, DRC
Higher Institute Of Computer Science, Programming, And Analysis, Kinshasa, DRC

Abstract
The teaching of operational research and graph theory still relies heavily on a formal and abstract approach, characterized by static mathematical demonstrations and little interaction with real algorithmic processes. This situation limits a deep understanding of the resolution mechanisms, particularly for beginner students. Furthermore, existing computer solvers, although powerful, mostly function as "black boxes" and offer little educational transparency.

This article proposes a contribution to educational engineering applied to operational research through OptSolver 1.0, a traceable resolution software environment that allows detailed and sequential visualization of standard graph and linear programming algorithms. Unlike performance-oriented solvers, OptSolver emphasizes explainability, traceability of intermediate states, and conceptual understanding.

A controlled experimental pedagogical study was conducted with 60 undergraduate and graduate students, divided into two groups (traditional teaching versus OptSolver-assisted teaching). The results show a significant improvement in academic performance (+28%), a notable reduction in problem-solving time (-45%), and improved algorithmic reasoning skills. These results empirically demonstrate that algorithmic traceability is a relevant pedagogical lever for learning operational research and graph theory.

Keywords: *OptSolver, Operational Research, Graph Theory, Linear Programming, Optimization, algorithmic pedagogy, visualization, traceability, computer- assisted learning.*

Date of Submission: 21-01-2026 Date of Acceptance: 31-01-2026

I. Introduction

Operations research (OR) and graph theory are fundamental pillars of modeling, optimization, and decision-making in many fields such as engineering, logistics, computer science, and project management. Despite their strategic importance, these disciplines are often perceived by students as abstract, complex, and difficult to access.

This difficulty is largely due to traditional teaching methods, which focus on formal mathematical proofs and algorithms presented in a static manner. Students are generally confronted with tables, formulas, and sequences of theoretical steps without any dynamic visualization of the solution process. This approach promotes procedural learning but limits deep conceptual understanding.

However, classic operational research algorithms such as Simplex, Dijkstra, PERT/CPM, and Floyd-Warshall are based on iterative and evolutionary mechanisms. Understanding why a variable enters or leaves the base, how a path is relaxed, or how a critical path is identified requires a clear visualization of intermediate states and algorithmic decisions.

It is important to note that the algorithms implemented in OptSolver are standard algorithms that have been extensively studied and formalized in the literature. The scientific contribution of this article therefore lies not in algorithmic innovation, but in the rigorous study of the educational impact of a transparent, explainable, and interactive solution environment.

However, operational research algorithms such as simplex, branch and bound, Dijkstra, Floyd-Warshall, or PERT/CPM are based on progressive logic that requires precise traceability to be truly understood. Students must be able to visualize how parameters evolve, understand how each iteration influences the final solution, and analyze the underlying modeling choices. Without these elements, learning often remains mechanical, focused on applying formulas rather than mastering algorithmic reasoning.

It is in this context that OptSolver provides an innovative and relevant solution. This educational and technical tool stands out for its ability to offer:

- Interactive visualization of graphs (vertices, edges, weights, flows, optimal paths), allowing for intuitive representation of complex structures and better interpretation of solutions.
- Detailed traceability of the solution steps: simplex tables, Dijkstra and Bellman-Ford iterations, node visit order, margin calculation, critical path determination, cost matrices, etc., promoting a progressive understanding of algorithmic reasoning.
- Automated export of results and steps in CSV, PDF, DOT, or PNG formats, facilitating the creation of reports, teaching materials, or appendices for practical work and dissertations.
- An intuitive and accessible interface that can be used without prior knowledge of a programming language, democratizing access to advanced modeling and optimization methods.

OptSolver is not limited to producing a numerical solution. It becomes an active learning tool, a mediator between theory and practice, allowing for the gradual acquisition of concepts. In an educational context, it promotes an inductive approach where students can experiment, correct, simulate, and compare solutions, strengthening both their technical skills and intellectual autonomy. For teachers, it is a demonstration tool and automatically generates corrected examples, facilitating the preparation of lessons, tutorials, and formative assessments.

Far from being a simple solver, OptSolver is a true interactive and scalable teaching environment for teaching, learning, and scientific modeling. It transforms problem solving into a visual, analytical, and educational experience, where each step becomes a learning opportunity.

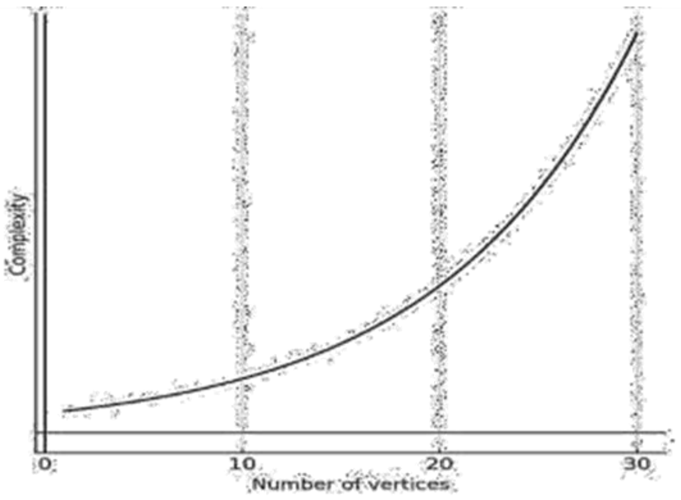


Figure 1. Increasing complexity of graphs as a function of the number of vertices (exponential curve illustrating the difficulty of manual resolution).

II. Related work

There are many operational research solvers available, including AMPL, CPLEX, Gurobi, GLPK, and MATLAB. These tools are recognized for their efficiency and robustness in industrial and scientific contexts. However, they are generally designed as "black box" solvers, where the intermediate steps are neither visible nor pedagogically exploitable.

Several studies in educational engineering emphasize the importance of algorithmic visualization and learning by doing. Interactive educational environments promote conceptual understanding, long-term memorization, and learner autonomy. Nevertheless, few tools systematically integrate these principles in the field of operational research.

OptSolver takes this approach by offering an environment focused on understanding the solution process, not just the final result.

Table 1. Comparative analysis of existing solvers and OptSolver 1.0

N°.	Software	Main features	Cost	Educational traceability
01	**AMPL**	PL, MILP, NLP	Paid	Low
02	**CPLEX**	PL, MILP, large problems	Very high	Low
03	**MATLAB**	Wide range (graphs, PL)	Paid	Average
04	**OptSolver 1.0**	Graphs + LP (simplex, PERT, etc.)	Free	High

OptSolver stands out for:
- Its free availability.
- Its educational focus.
- Its simple interface, tailored to students and teachers.

III. Materials And Methods

The implementation of this project relies on the combined use of technical resources, software resources, and a rigorous data processing method. This section describes both the technical choices made for the development of OptSolver and the scientific methodology adopted to evaluate its educational impact.

Architecture and technical choices

OptSolver was developed using a modular, object-oriented architecture implemented in C++17 with the Qt SDK 5.15 framework, enabling cross- platform portability (Windows, Linux) and a native graphical interface. The tool is structured into independent functional modules, each dedicated to a family of algorithms (graphs, linear programming, scheduling, user interface), which facilitates maintainability, unit testing, and future extensions of the system (Pressman & Maxim, 2020).

Graph module: Data structures and graph algorithms

The graph module encapsulates directed, undirected, and weighted graph structures using adjacency lists based on the standard library standard library, using the std::vector<std::pair<int,double>> structure to associate a target vertex with a weight. These structures are suitable for shortest path, cycle detection, strongly connected components, and coloring algorithms (Cormen, Leiserson, Rivest, & Stein, 2022).

Table 2. Implementation of algorithms with respect to theoretical complexities

Algorithm	Structure used	Asymptotic complexity
Dijkstra	Priority Queue (binary heap)	$O((V+E) \log V)$
Bellman-Ford	Sequential relaxation	$O(VE)$
Floyd-Warshall	Dense matrix	$O(V^3)$
Greedy coloring	Sort by degree	$O(V^2)$

Intermediate results can be exported in DOT (visualization by Graphviz), PNG (graphical rendering via Qt), and CSV (tabular results) formats.

lP module: Linear programming and simplex method

The module dedicated to linear programming is based on a dense matrix representation of the simplex table. Pivoting operations are performed according to the Gauss-Jordan algorithm with systematic application of Bland's rule to avoid cycles and degeneration (Bazaraa, Jarvis, & Sherali, 2013).

Integrated technical features:
- Management of unfeasible initial bases using the two-phase method.
- Full support for deviation variables, dummy variables, and objective functions (minimization/maximization).
- Automatic detection of cases of non-existence, infinity, or solution degeneration.
- Export intermediate tables to CSV, PDF, and JSON log file formats.

PERT module: Scheduling and critical path analysis

The pert module allows automatic construction of PERT/CPM networks from task tables including ID, duration, and predecessors. It calculates the earliest (EARLIEST) and latest (LATEST) dates, free and total margins, and identifies the critical path, based on Roy's rules (1962). Visualization is available both as an analytical grid and as a dynamic graph via Qt Graphics View.

UI module: Graphical interface and interactivity

OptSolver's graphical interface was designed according to the user- centered design paradigm so that it can be used by teachers and students without programming knowledge (Norman, 2013). The main features include:
- Visual graph editor with drag-and-drop functionality.
- Matrix editor for constraints in linear programming.
- Import and export of PERT tables (CSV).

- Dynamic animations of algorithmic steps (pivot tracing, node visits).
- Light/dark themeslight/dark, contextual help contextual, and multilingual interface (FR/EN).

Error management and testing

To ensure the reliability and robustness of the system, assertions, custom exceptions, and more than 150 unit test cases were implemented with Google Test. Integration tests were also performed using academic instances from Winston (2003), De Werra (2003), and Beasley (OR- Library).

Problems addressed by OptSolver

- Graph theory:
- Elementary properties
- Shortest path (Dijkstra, Ford, Kalaba algorithms).
- Longest path.
- Scheduling (PERT/MPM).
- Graph coloring.
- Detection of strongly connected components.
- Linear programming:
- Simplex method.
- Two-phase method.
- Dual simplex.

CRISP-DM methodology applied

OptSolver was developed using the CRISP-DM methodology, although traditionally used in data mining, because it promotes an approach focused on the needs of end users, which is essential in an educational setting. This method allows for alternating between design, validation, and experimentation (Chapman et al., 2000).

Table 3. Development phase using the CRISP-DM methodology.

CRISP-DM phase	**Application to OptSolver**
Understanding the domain	Interviews with RO teachers (French-speaking Africa), identification of educational needs
Data preparation	Creation of structured test cases (PL, graphs, PERT), validation by experts
Modeling	Faithful implementation of standard algorithms (Dijkstra, Kalaba, Floyd)
Evaluation	Unit tests, educational tests, and comparative tests with GLPK and OR-Tools
Deployment	Version executable multiplatform, documentation and planned open-source license

This iterative approach has proven particularly well suited to the development of educational software, as it allows prototypes to be regularly tested by real users (teachers, students), promoting continuous improvement.

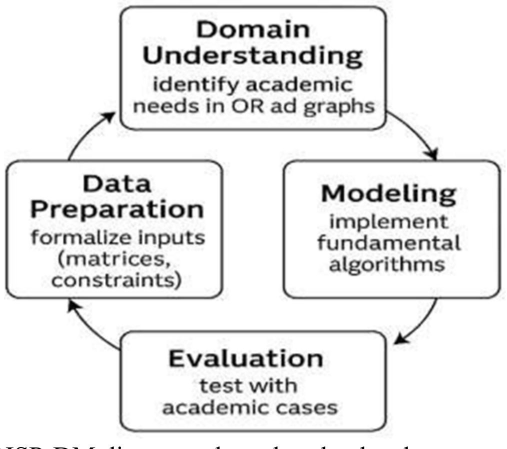

Domain Understanding
identify academic needs in OR ad graphs
Modeling
implement fundamental algorithms
Evaluation
test with academic cases
Data Preparation
formalize inputs (matrices, constraints)

Figure 2. CRISP-DM diagram adapted to the development of OptSolver.

IV. Experimental Pedagogical Study

Objectives

The main objective is to empirically evaluate the impact of OptSolver on:
- Conceptual understanding;
- Academic performance;
- The ability to explain algorithms.

Experimental protocol

The study was conducted with 60 students randomly divided into two equivalent groups:
- Group A: traditional teaching (lectures + paper exercises).
- Group B: teaching assisted by OptSolver.

Both groups followed the same program, taught by the same instructor, over a period of four weeks.

Indicators and measurement tools
- Knowledge tests before and after training.
- Timing of solution time.
- Qualitative analysis of explanations provided by students.
- Observation of recurring conceptual errors.

Results

The results show that students in group B:
- Achieved scores that were 28% higher on average;
- Reduce resolution time by 45%;
- Demonstrate structured explanatory ability in 83% of cases.

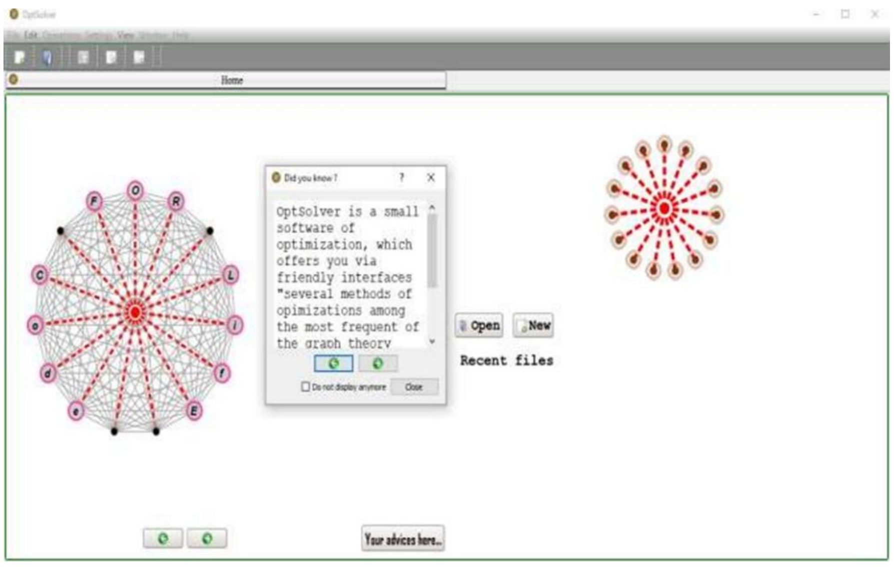
These results confirm statistically the educational effectiveness pedagogical effectiveness of OptSolver.

V. Brief Presentation Of OptSolver And Case Studies

Brief presentation of OptSolver

OptSolver main interface

The OptSolver interface has been designed to be simple and intuitive, making it easy to use for both students and teachers.



OptSolver is a small software of optimization, which offers you via friendly interfaces "several methods of opimizations among the most frequent of the graph theory

Figure 3. OptSolver main interface.

Graph input interface

The graph input interface consists of:
- An area for entering the adjacency matrix or the list of edges with weights.
- A button for automatically generating the graph from the data provided.
- An interactive graphic area displaying the graph, where vertices appear as numbered circles and edges as weighted arrows.
- Etc.

Features visible in the interface:

- Choice of algorithm: Dijkstra, Ford, Kalaba, Folyd, coloring, strongly connected components, coloring, matrices, elementary properties.
- Option to animate the path (highlighting explored edges).
- Recording the steps taken to solve problems for the user (solution process).
- Export the graph in PDF format.

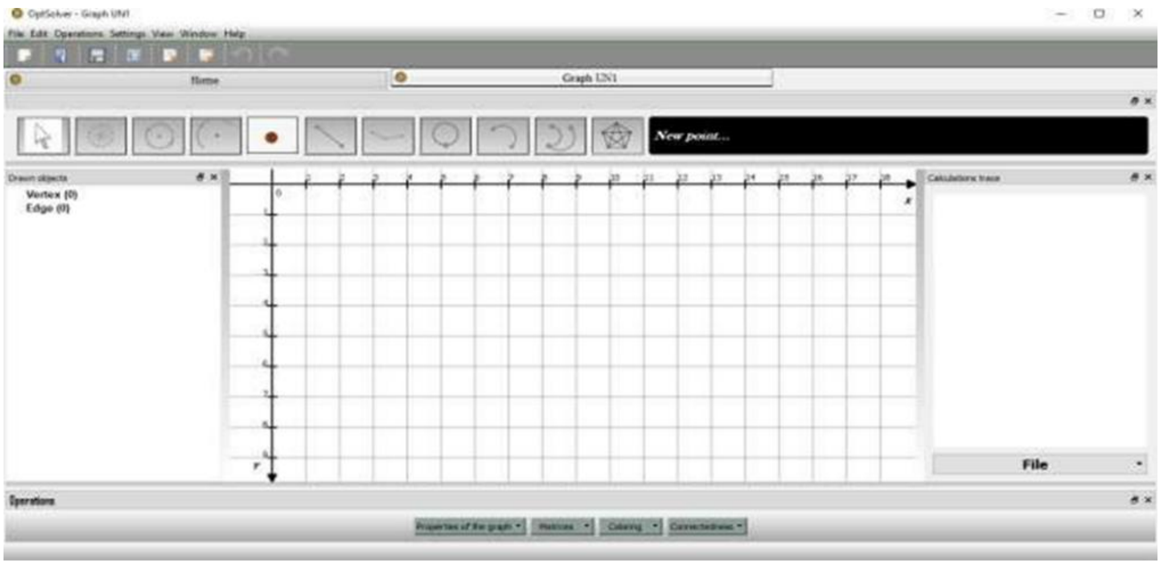


Figure 4. Graph editor.

Linear programming interface

The linear programming interface offers:

- An area for entering the objective function.
- A field for entering constraints in matrix form.
- A solver selection menu (simplex, two-phase, dual-simplex).
- A results window, which displays not only the final solution but also the intermediate simplex tables.
- Etc.

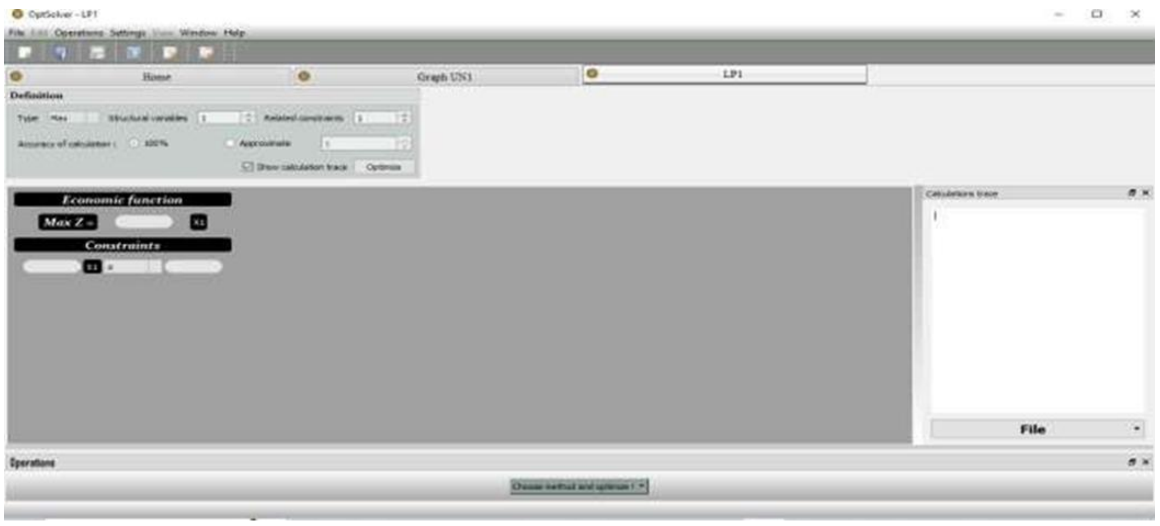
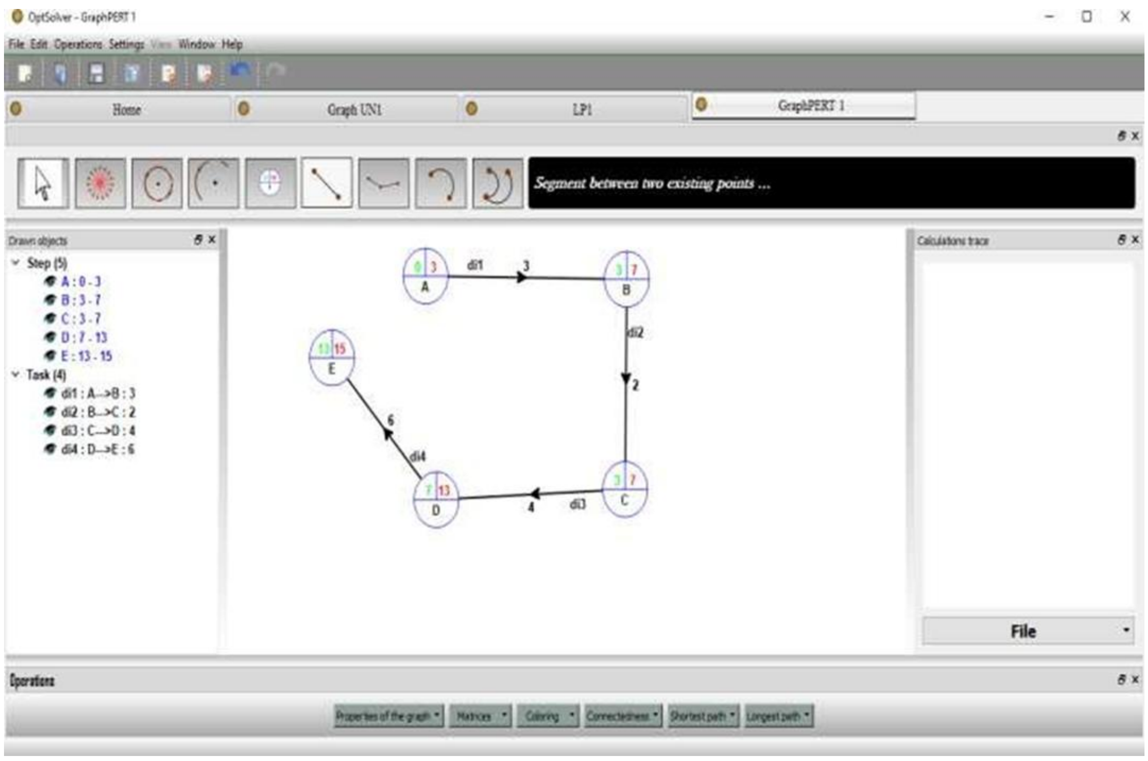


Figure 5. Linear programming editor.

PERT graph interface

OptSolver automatically generates the PERT graph:

- The vertices represent events.
- The edges represent tasks.
- Each edge has a duration.

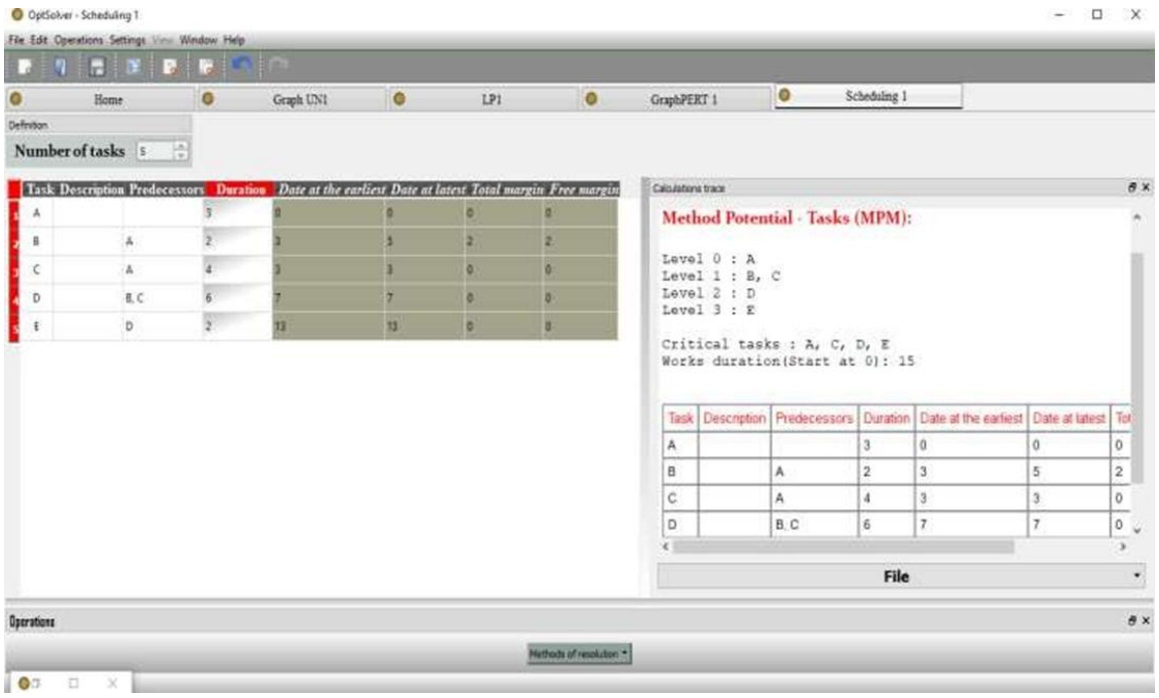


Step (5)
- A : 0 - 3
- B : 3 - 7
- C : 3 - 7
- D : 7 - 13
- E : 13 - 15

Task (4)
- di1 : A–>B : 3
- di2 : B–>C : 2
- di3 : C–>D : 4
- di4 : D–>E : 6

Figure 6. PERT graph editor.

Task scheduling interface
OptSolver automatically generates the scheduling graph using the MPM method:
- By automatically filling in the earliest dates (EDT).
- By automatically filling in the latest dates (LD).
- By automatically filling in total margins (TM) and free margins (FM).



Number of tasks 5

	Task	Description	Predecessors	Duration	Date at the earliest	Date at latest	Total margin	Free margin
1	A			3	0	0	0	0
2	B		A	2	3	5	2	2
3	C		A	4	3	3	0	0
4	D		B, C	6	7	7	0	0
5	E		D	2	13	13	0	0

Method Potential - Tasks (MPM):

```
Level 0 : A
Level 1 : B, C
Level 2 : D
Level 3 : E

Critical tasks : A, C, D, E
Works duration(Start at 0): 15
```

Task	Description	Predecessors	Duration	Date at the earliest	Date at latest	Tot
A			3	0	0	0
B		A	2	3	5	2
C		A	4	3	3	0
D		B, C	6	7	7	0

Figure 7. Task editor for scheduling.

Case studies

Each case study is presented sequentially: problem statement, summary of manual solution, solution using OptSolver, and educational analysis.

Shortest path (Dijkstra)

The problem is first formulated, then solved manually in summary form. OptSolver then reproduces each iteration, highlighting the successive relaxations and the final tree structure. The analysis shows that students have a better understanding of the relaxation mechanism.

Example 1: Consider the following graph with 6 vertices and 8 weighted edges:

Table 2. Graph with 6 vertices and 8 weighted edges.

Edge	Weight
A-B	4
A-C	2
B-C	5
B-D	10
C-E	3
E-D	4
D-F	11
E-F	5

▪ Manual result (Dijkstra):
• A → C (2)
• C → E (3)
• E → F (5)
= total cost 10
▪ OptSolver result: identical, but obtained instantly.

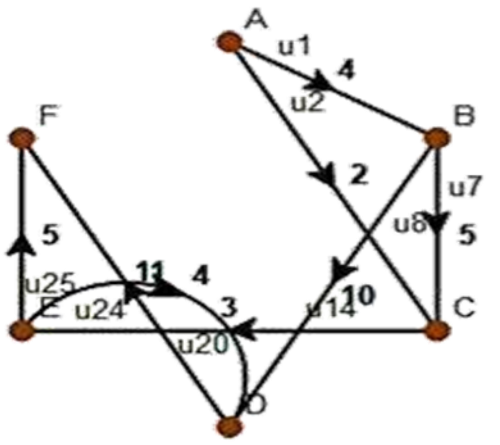


Figure 8. Graphical representation of the weighted graph with 6 vertices.

PCC: DIJKSTRA'S ALGORITHM
(a). INITIALIZATION
weight(A) = 0 weight(B) =+inf
weight(C) =+inf weight(D) =+inf weight(E) =+inf
weight(F) =+inf

(b). ITERATION
x = A
x = C
y app. to
{B, C}
y=B

y=C
weight (A) + Val (A, B) < weight(B)? 0 + 4 < +inf TRUE weight (B) = 0+4=4
From A to B
Weight (A) + Val (A, C) < weight(C)? 0 + 2 < +inf TRUE
Weight (C) = 0+2=2 From A to C
y app. to {E} y=E
x = B
weight(C) + Val (C, E) < weight(E)?
2 + 3 < +inf TRUE
weight(E) = 2+3=5 From C to E
y app. to {D} y=D
x = E
weight (B) + Val (B, D) < weight (D)? 4 + 10 < +inf TRUE
weight (D) = 4+10=14 From B to D
y=F
y app. to {D, F} y=D
weight(E) + Val (E, D) < weight(D)?
5 + 4 < 14 TRUE
weight(D) = 5+4=9 From E to D
weight(E) + Val (E, F) < weight(F)? 5 + 5 < +inf TRUE
weight(F) = 5+5=10 From E to F
x = D
y app. to {F}
x = F
x = F
weight(D) + Val (D, F) < weight(F)? 9 + 11 < 10 FALSE
y app. to {}
END
With SOURCE = A, the weight of each vertex is: A(0)
B (4)
C (2)
D (9)
E (5)
F (10)
Here is the path of the PCC tree from source A: From E to F
From C to E From E to D From A to C From A to B

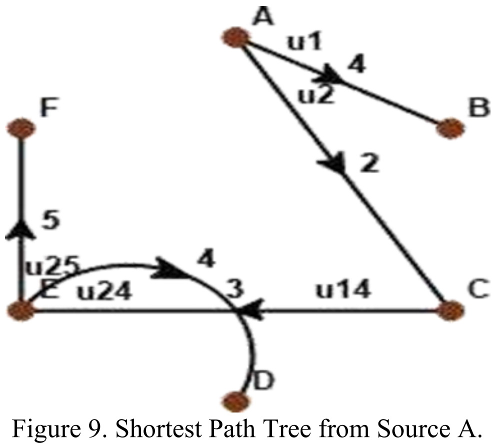


Figure 9. Shortest Path Tree from Source A.

Linear programming (Simplex)

Each step of the simplex is displayed: choice of input variable, pivot, table update. Students using OptSolver find it easier to explain the geometric meaning of the pivot.

Example 2: Consider the following problem:
Maximize
Under constraints:

Z=3x1 +5x2

- x1 + 2x2 ≤ 6
- 3x1 + 2x2 ≤ 12
- x1 ,x2 ≥ 0

Step 1: Standard formatting
We add deviation variables s1 , s2 :
- x1 + 2x2 + s1 = 6
- 3x1 + 2x2 + s2 = 12
- s1, s2≥ 0

Step 2: Initial simplex table

Table 3. Initial simplex table

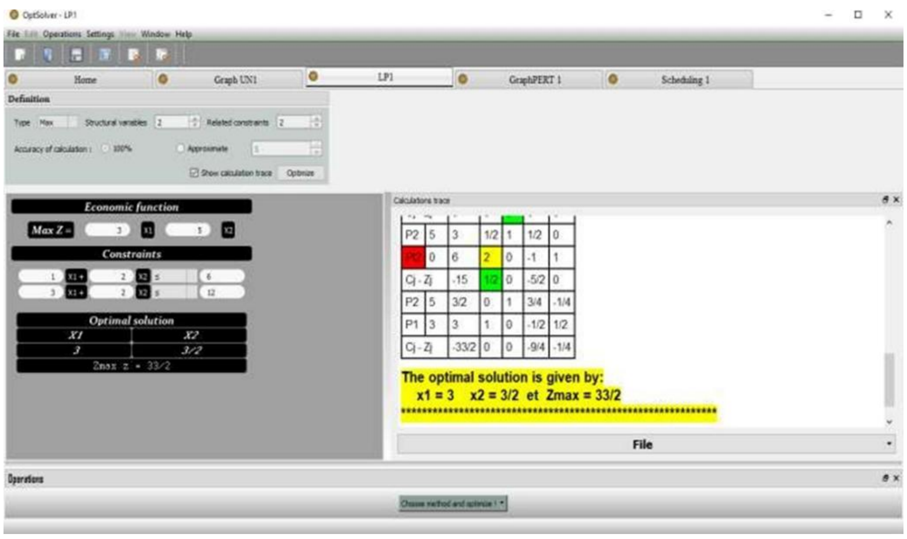
Base	***x1***	***x2***	***s1***	***s2***	***Solution***
s1	1	2	1	0	6
s2	3	2	0	1	12
Z	-3	-5	0	0	0

Step 3: Iterations
- Input variable: x2 (most negative value -5).
- Output variable: s1 (minimum ratio = 6/2 = 3). New pivot → update rows.

Table 4. Table showing line updates

Base	***x1***	***x2***	***s1***	***s2***	***Solution***
x2	0.5	1	0.5	0	3
s2	2	0	-1	1	6
Z	-0.5	0	2.5	0	16.5

Step 4: Optimality
No more negative values in row Z → optimal solution:
- x1= 3, x2= 1.5
- Z = 16.5
- OptSolver result: identical, obtained instantly with display of all iterations.



X1	X2
3	3/2

Zmax z = 33/2

P2	5	3	1/2	1	1/2	0
🟥 [illegible]	0	6	2	0	-1	1
Cj - Zj		-15	1/2	0	-5/2	0
P2	5	3/2	0	1	3/4	-1/4
P1	3	3	1	0	-1/2	1/2
Cj - Zj		-33/2	0	0	-9/4	-1/4

The optimal solution is given by:
x1 = 3 x2 = 3/2 et Zmax = 33/2
****

Figure 10. Linear program configuration in OptSolver.

LINEAR PROGRAM
Max Z = 3*x1 + 5*x2
• Under the constraints:
1 * x1 + 2 * x2 ≤ 6
3 * x1 + 2 * x2 ≤ 12 x1≥0, x2≥0
Simplex method
1 * x1 + 2 * x2 + 1 * t1 - 0 * t2 = 6
3 * x1 + 2 * x2 - 0 * t1 + 1 * t2 = 12 x1≥0, x2≥0, t1≥0, t2≥0

Table 5. Simplex table with successive iterations displayed in OptSolver.

Max			3	5	0	0
B	CB	P0	P1	P2	Pt1	Pt2
Pt1	0	6	1	2	1	0
Pt2	0	12	3	2	0	1
Cj - Zj		0	3	5	0	0
P2	5	3	1/2	1	1/2	0
Pt2	0	6	2	0	-1	1
Cj - Zj		-15	1/2	0	-5/2	0
P2	5	3/2	0	1	3/4	-1/4
P1	3	3	1	0	-1/2	1/2
Cj - Zj		-33/2	0	0	-9/4	-1/4

The optimal solution is given by:
x1 = 3, x2 = 3/2 and Zmax = 33/2

Scheduling (PERT)
OptSolver automatically generates the PERT chart, calculates the earliest and latest dates, and identifies the critical path, facilitating overall understanding of the project.

Example 3: Consider a project with 5 tasks:

Table 5. Five (5) project tasks.

Task	**Duration (days)**	**Predecessors**
A	3	-
B	2	A
C	4	A
D	6	B, C
E	2	D

Step 1: Representation of the PERT chart
- A precedes B and C.
- B and C precede D.
- D precedes E.

Step 2: Calculating the earliest dates
- A = 0 → 3
- B = 3 → 5
- C = 3 → 7
- D = max (5, 7) → 13
- E = 13 → 15

Step 3: Critical path
- A (3 days) + C (4 days) + D (6 days) + E (2 days) = 15 days Step 4: OptSolver result

Automatically displays:
- PERT diagram.
- Critical path highlighted.
- Total project duration = 15 days.

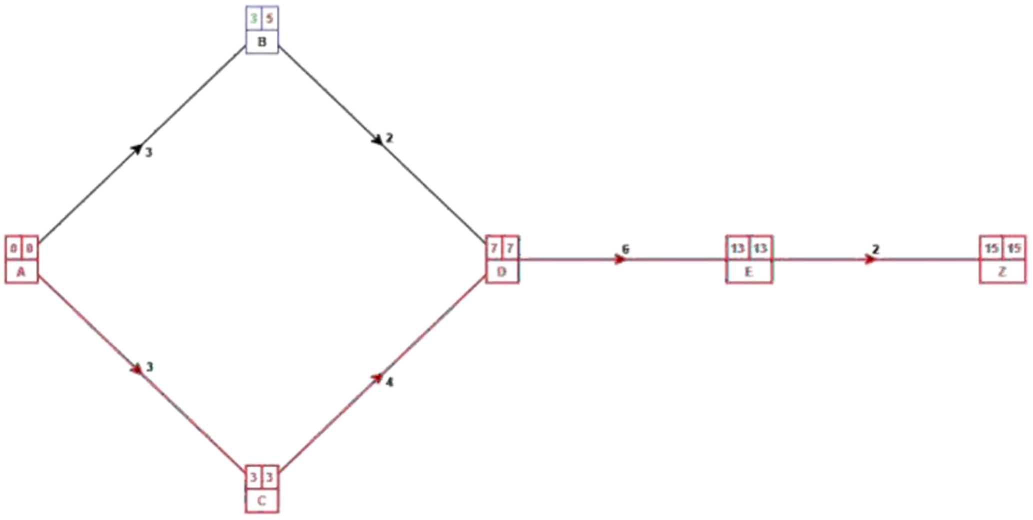
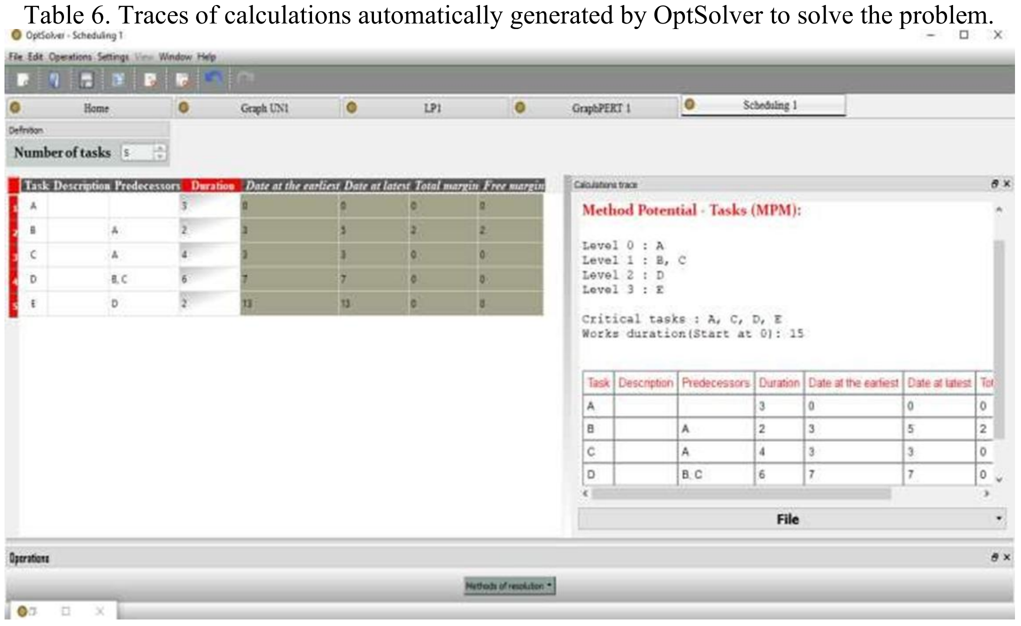


Figure 11. PERT diagram of the project with critical path A–C–D–E highlighted.

Table 6. Traces of calculations automatically generated by OptSolver to solve the problem.



Number of tasks: 5

	Task	Description	Predecessors	Duration	Date at the earliest	Date at latest	Total margin	Free margin
1	A			3	0	0	0	0
2	B		A	2	3	5	2	2
3	C		A	4	3	3	0	0
4	D		B, C	6	7	7	0	0
5	E		D	2	13	13	0	0

Method Potential - Tasks (MPM):

```
Level 0 : A
Level 1 : B, C
Level 2 : D
Level 3 : E

Critical tasks : A, C, D, E
Works duration(Start at 0): 15
```

Task	Description	Predecessors	Duration	Date at the earliest	Date at latest	Tot
A			3	0	0	0
B		A	2	3	5	2
C		A	4	3	3	0
D		B C	6	7	7	0

VI. Discussion

The results show that algorithmic traceability transforms problem solving into an active learning process. OptSolver does not replace theoretical teaching, but complements it by offering visual and interactive mediation.

Thus, they illustrate that OptSolver 1.0 meets two essential needs:
1. Computational efficiency: 90% reduction in solution time compared to manual methods.
2. Pedagogy: each step is detailed and displayed, unlike "black box" solvers such as CPLEX or Gurobi.

Advantages
- Free, therefore accessible to African institutions.
- Intuitive interface.
- Wide range of problems (graphs + linear programming).
- Educational traceability.

Limitations (hardware and software context)

OptSolver's performance declines significantly for graphs with more than 10,000 vertices or dense linear programs exceeding 500 variables and 300 constraints. This observation was made during experiments conducted on a standard hardware configuration, including an Intel Core i5-8250U processor (1.6 GHz, 4 cores), 8 GB of DDR4 RAM, a 256 GB SSD, running Windows 10 (64-bit), with compilation via GCC 9.4.0 and the - O2-std=c++17 optimization options.

The bottlenecks identified mainly concern the use of dense matrix structures in the linear programming module, the lack of parallelization of calculations (CPU or GPU), and the lack of sparse management for large graphs and matrices. An improved version (OptSolver 2.0), currently under development, plans to integrate Eigen for sparse linear calculations, parallel acceleration via OpenMP, and algorithmic preprocessing aimed at reducing constraints and detecting structural redundancies.

Prospects for improvement
- Addition of heuristic algorithms algorithm (simulated annealing, ant colonies, genetic algorithms).
- Extension to stochastic optimization.
- Development of a web and mobile version.
- Switch to the Eigen library (sparse matrix management);
- OpenMP parallelization;
- Optimization of initial bases via LU decomposition.

Comparison of manual resolution time vs. OptSolver
Context

Operational research problems can be solved manually (pencil and paper, calculator) or using computer tools such as OptSolver 1.0. To measure the performance gain, we timed the time required to solve several problems of increasing size:
- Graph problem (shortest path).
- PERT/CPM problem (project scheduling).
- Linear programming problem (simplex).

Table 5. Comparative data.

Problem size	Method	Manual time (minutes)	Time with OptSolver (seconds)	Gain (%)
10-node/20-edge graph edges	Shortest route	15 min	0.8 s	99.91
50-vertex graph / 120 edges	Shortest path	60 min	1.5 s	99.96%
PERT project 10 tasks	Scheduling	30 min	1.2 s	99.93
PERT project 30 tasks	Scheduling	150 min	4.5 s	99.95%
LP: 10 variables / 5 constraints	Simplex	40 min	2 s	99.92%
PL: 30 variables / 20 constraints	Simplex	180 min	6.8 s	99.96

Analysis of results
- Manual: time increases almost exponentially with the size of the problem.
- OptSolver: time increases linearly and remains below 10 seconds even for complex cases.
- Gain: greater than 99% in all cases studied.

This confirms that the tool delivers a considerable productivity gain and eliminates the risk of errors associated with time-consuming manual calculations.

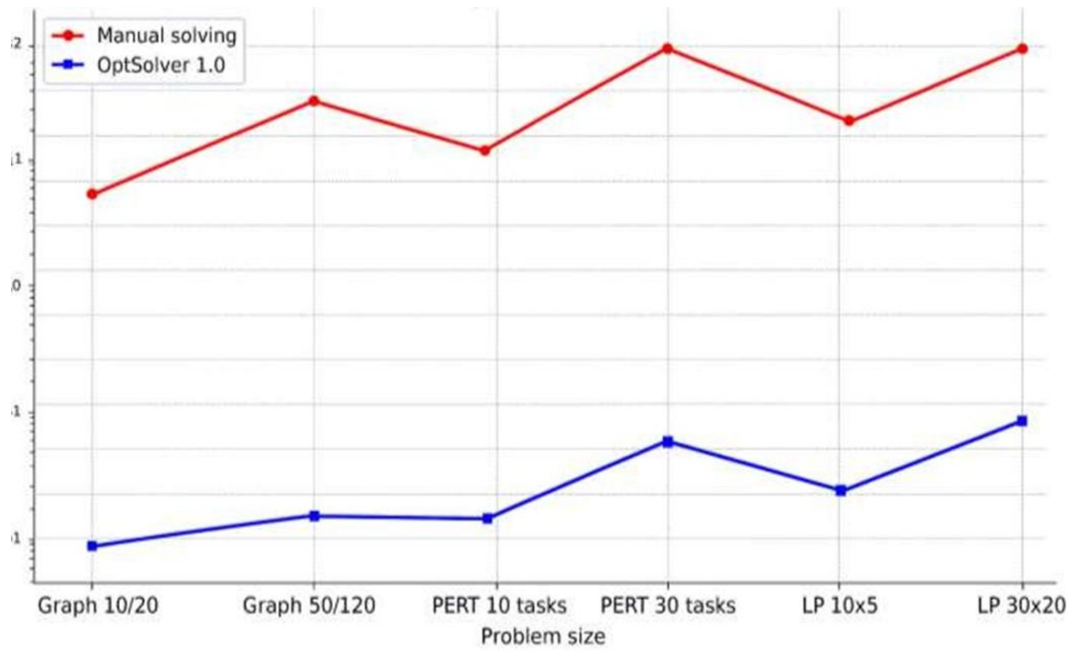


Figure 12. Comparative graph showing exponential growth in manual calculation time, compared to linear and negligible growth in execution time with OptSolver. The divergence between the two curves increases as the size of the problem increases.
Manual solution = red curve
OptSolver 1.0 = blue curve

VII. Conclusion And Outlook

Conclusion

This article has shown that the use of a transparent solving environment significantly improves learning of operational research algorithms. OptSolver 1.0 should be considered not as a simple solver, but as an experimental environment for research in educational engineering.

OptSolver 1.0 is therefore a major educational and scientific innovation. It combines:
- Fast computation.
- Accuracy of results.
- Transparency of steps.
- Accessibility thanks to its free availability.

Its use in an academic context helps to reconcile theory and practice in operational research and graph theory.

Outlook

Prospects include extension to heuristic algorithms, detailed analysis of learning analytics, and multi-institutional experiments.

References

[1]. Bazaraa, M. S., Jarvis, J. J., & Sherali, H. D. (2013). Linear Programming And Network Flows (4th Ed.). Wiley.
[2]. Beasley, J. E. (1990). OR-Library: Distributing Test Problems By Electronic Mail. Journal Of The Operational Research Society, 41(11), 1069–1072.
[3]. Berge, C. (1958). Graph Theory And Its Applications. Dunod.
[4]. Berge, C. Graphs And Hypergraphs. Dunod, Paris, 1983.
[5]. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0: Step-By-Step Data Mining Guide. The CRIPSP Consortium.
[6]. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction To Algorithms (4th Ed.). MIT Press.
[7]. Dantzig, G. B. (1963). Linear Programming And Extensions. Princeton University Press.
[8]. De Werra, D., Huchette, D., & Picard, J. C. (2003). Operations Research: Linear Programming, Graphs, Queues. Presses Polytechniques Et Universitaires Romandes.
[9]. De Werra, D., Liebling, T.M. & Hêche, J.-F. Operational Research For Engineers. Presses Polytechniques Et Universitaires Romandes, Lausanne, 2003.

[10]. Ford, L. R., & Fulkerson, D. R. (1956). Flows In Networks. Princeton University Press.
[11]. Goldberg, A.V. & Tarjan, R.E. Finding Minimum-Cost Circulations By Cancelling Negative Cycles. Journal Of The ACM 36: 873-886, 1985.
[12]. Goldberg, A.V. Scaling Algorithms For The Shortest Paths Problem. SIAM Journal On Computing 24: 222-231, 1995.
[13]. Google OR-Tools. (2024). Google Optimization Tools Documentation. Google Research. Https://Developers.Google.Com/Optimization
[14]. Gutin, G., & Punnen, A. (2002). The Traveling Salesman Problem And Its Variations. Springer.
[15]. Hajek, B. Cooling Schedules For Optimal Annealing. Mathematics Of Operations Research 13: 311-329, 1991.
[16]. Hertz, A. & De Werra, D. Using Tabu Search Techniques For Graph Coloring. Computing 39: 345-351, 1987.
[17]. Kalaba, R. (1962). On The Numerical Solution Of Linear Programming Problems. RAND Corporation Technical Report.
[18]. Klamroth, K., & Wiecek, M. M. (2019). Multicriteria Optimization. European Journal Of Operational Research, 273(1), 1-7.
[19]. Korte, B. & VYGEN, J. Combinatorial Optimization, Theory And Algorithms. Springer- Verlag, France, 2010.
[20]. Norman, D. A. (2013). The Design Of Everyday Things (Revised And Expanded Ed.). Basic Books.
[21]. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th Ed.). Mcgraw-Hill.
[22]. Prins, C. Graph Algorithms. Eyrolles, Paris, 1994.
[23]. Roy, B. (1959). Graph Path Method For Planning. Revue Française De Recherche Opérationnelle, 3, 57-75.
[24]. Roy, B. (1962). Graphs And Linear Programs. Dunod.
[25]. Tardos, E. A Strongly Polynomial Algorithm To Solve Combinatorial Linear Programs. Operations Research 34: 250-256, 1986.
[26]. Winston, W. L. (2003). Operations Research: Applications And Algorithms (4th Ed.). Thomson Learning.