An Improvement of Backtracking Algorithms for Combinatorial Optimization Problems

Claudiu Pozna[1,2], Bogdan Sibișan[1], Radu-Emil Precup[3,4]

[1] Dept. of Automation and Information Technology, Transilvania University of Brasov, Brasov, Romania

[2] Faculty of Engineering Sciences, Széchenyi István University, Győr, Hungary

[3] Dept. of Automation and Applied Informatics, Politehnica University of Timisoara, Timisoara, Romania

[4] Center for Fundamental and Advanced Technical Research, Romanian Academy – Timisoara Branch, Timisoara, Romania

e-mail: cp@unitbv.ro, bogdan.trasnea@unitbv.ro, radu.precup@upt.ro

Abstract: This paper introduces an improvement approach of the classical backtracking algorithm tailored for solving complex combinatorial problems. The proposed approach restructures the search space by grouping solution sets into meta-sets and applying continuity criteria early in the search process, leading to a significant reduction of the number of iterations required to reach valid solutions. The effectiveness of the approach is demonstrated through case studies involving Sudoku and Kakuro puzzles, where it outperforms traditional backtracking in terms of computational efficiency and solution accuracy. The results show that the proposed approach offers a scalable and practical enhancement for combinatorial optimization tasks.

Keywords: backtracking algorithms; combinatorial optimization; constraint satisfaction problems; search space reduction.

1 Introduction

Backtracking algorithms are fundamental in solving combinatorial and constraint satisfaction problems. They work by incrementally constructing candidate solutions and discarding paths that violate constraints. While conceptually simple and widely applicable, classical backtracking can be inefficient for large or complex problem spaces due to its exhaustive search nature, often resulting in high computational overhead.

Various enhancements have been proposed to improve backtracking, such as pruning strategies, heuristics, and hybrid methods that integrate dynamic programming or greedy algorithms. However, these still typically explore individual elements iteratively, which limits scalability and can lead to thousands or millions of iterations for moderately sized problems.

This paper introduces a novel approach that improves the efficiency of the classical backtracking algorithms by restructuring the search process. The proposed approach involves grouping solution candidates into meta-sets and applying continuity criteria early in the traversal. By shifting the focus from individual elements to structured subsets of the solution space, the suggested approach formulated as an algorithm significantly reduces the number of iterations required. This not only improves performance but also enhances the algorithm's ability to handle problems with complex interdependencies and constraints.

To validate the effectiveness of the proposed approach, it is applied to two well-known combinatorial puzzles, Sudoku and Kakuro, as representative case studies due to their structured constraints and varying levels of complexity. Comparative analysis with classical backtracking demonstrates substantial gains in computational efficiency, with the improvement approach requiring fewer iterations and exhibiting faster convergence. The results suggest that this approach offers a scalable and practical enhancement for solving a wide range of combinatorial optimization problems.

The rest of the paper is structured as follows: the literature review is discussed in the next section, the problem definition and the classical backtracking are described in Section 3, the proposed improvement approach is presented in Section 4, the case studies are discussed in Section 5, and the conclusions are outlined in Section 6.

2 Literature Review

Backtracking algorithms have long been a fundamental component of algorithmic problem-solving, particularly in the context of combinatorial search and constraint satisfaction problems. These algorithms systematically explore the solution space by incrementally building candidates and abandoning paths that violate constraints. Foundational works such as Knuth's The Art of Computer Programming provide a rigorous and comprehensive treatment of combinatorial algorithms, including backtracking, laying the groundwork for decades of research and application [1].

Cormen et al. present in [2] formal definitions and examples of backtracking in constraint satisfaction and search problems. Kleinberg and Tardos discuss in [3] how backtracking can be combined with greedy heuristics and dynamic programming to solve complex optimization problems. Weiss provides in [4] detailed examples of backtracking in C++ environments, emphasizing performance and memory efficiency. Sedgewick and Wayne offer in [5] practical insights into the implementation and performance of backtracking in real-world scenarios. Papadimitriou and Steiglitz focus in [6] on the complexity and efficiency of backtracking in optimization tasks, particularly in problems involving large search spaces. Papadimitriou outlines in [7] the theoretical limits of backtracking approaches, especially in NP-complete domains, and highlights the importance of pruning strategies. Aho et al. conduct in [8] a rigorous analysis of algorithmic strategies, including backtracking, within the broader context of computational complexity. Cormen offers in [9] an accessible entry point for understanding the core principles of backtracking, making it suitable for both students and practitioners. Harel and Feldman explore in [10]

the conceptual elegance of backtracking while acknowledging its practical challenges, such as exponential time complexity in worst-case scenarios. Skiena presents in [11] a wide array of real-world problems solvable via backtracking, from puzzle solving to bioinformatics. Sedgewick and Lafore offer in [12] and [13], respectively, language-specific implementations, emphasizing modularity and object-oriented design. Levitin adds in [14] a pedagogical perspective, emphasizing the role of backtracking in algorithmic design and its relationship to other paradigms such as divide-and-conquer and dynamic programming, while Goodrich and Tamassia extend in [15] these ideas to practical domains, demonstrating how backtracking can be adapted to solve scheduling, routing, and resource allocation problems efficiently. Roughgarden compares in [16] backtracking with greedy and dynamic programming approaches, offering insights into when each method is most effective. Mehlhorn and Sanders provide in [17] a pragmatic toolkit for evaluating and optimizing algorithmic performance, including techniques for reducing redundant computations and improving memory usage. Levitin and Levitin offer engaging examples that challenge readers to apply backtracking creatively in puzzle-solving contexts. Aziz and Lee highlight in [19] the relevance of backtracking in technical interviews, where it is often used to solve problems involving permutations, combinations, and constraint satisfaction. Bentley rounds out in [20] literature with insightful case studies that demonstrate how backtracking can be used to write elegant and efficient code in practice.

The literature on backtracking algorithms highlights their evolution from foundational theory to advanced optimization techniques and practical implementations. While classical backtracking provides a robust framework for exhaustive search, its efficiency diminishes with increasing problem complexity. This has spurred extensive research into pruning strategies, heuristic enhancements, and hybrid methods that aim to preserve flexibility while improving scalability.

Building on these insights, the proposed method in this paper introduces structural improvements that enhance performance and reduce iteration overhead. By leveraging established principles and addressing the limitations of traditional approaches, this optimization contributes meaningfully to the ongoing development of backtracking in algorithmic design.

3 Problem Definition and Classical Backtracking

The problem addressed in this paper involves identifying a subset $\{x^*\}$ of the Cartesian product of several finite sets $S = S_1 \times S_2 \times, \ldots, S_n$, such that a given objective function $g: S \to \{0, \quad 1\}$ is satisfied. The solution space S is generated by the finite sets S_i, each with cardinality $|S_i| = s_i \quad i = 1, \ldots, n$. A generic element of space is $x = [x_1 \quad \ldots \quad x_{n-1} \quad x_n]^T$, $x_i \in S_i$.

The solutions to the problem are denoted using the superscript *: $x^* = [x_1^* \quad \ldots \quad x_{n-1}^* \quad x_n^*]^T$. In certain cases, some components of the solution vector x* are predefined and denoted with the superscript ^{0}:

$$\exists i, \quad x^*(i) = x_i^0, \quad x_i^0 \in S_i \tag{1}$$

To guide the search process, a continuity criterion is associated with the objective function. This criterion determines whether the current partial solution x^k should be extended further: $c(x^k) = 1$, where $c: S^k \to \{0, \quad 1\}$, $S^k = S_1 \times S_2 \times, \ldots, S_k$, and $x^k = [x_1 \quad \ldots \quad x_{k-1} \quad x_k]^T$. It is noted that in many cases, the continuity criterion and the objective function are closely related, i.e. $g(x) \equiv c(x^n)$.

The search for the solution x* is modeled as a traversal through a tree-like graph (Figure 1). The graph starts from a virtual node V, which connects to nodes representing the variables in S_1, then to those in S_2, and so on, ending with variables in S_n. The solution is found by identifying combinations of variables that satisfy the condition in (1).

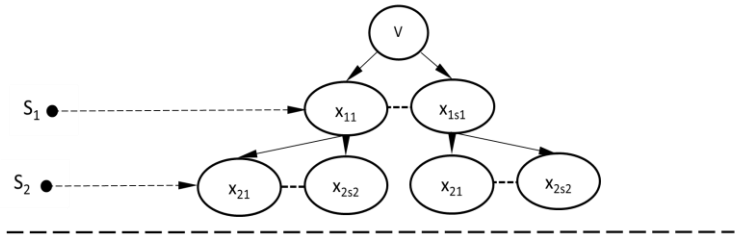


Figure 1
The graph (tree) associated with the problem

The classical backtracking algorithm constructs a candidate solution vector traversing a graph structure. The traversal proceeds sequentially through the sets S_i, for i = 1, 2, …, *n*, and applies a continuity criterion at each step to determine whether the current partial solution should be extended.

The iterative process is defined as follows:

- Step k = 1: Construct the partial solution $x^1 = [x_{1j}]$. For each j=1,…,s_1, check the continuity condition. If $c(x^1) = 1$, proceed to k =2, store the transition index j_1 = j, and set $x_1 \equiv x_{1j}$. If no element satisfies the condition, the algorithm terminates because there is no valid solution.
- Step k = 2: Extend the solution to $x^2 = [x_1 \; x_{2j}]^T$. For each j=1,…,s_2, check the continuity condition. If $c(x^2) = 1$, proceed to k=3, store the transition index j_1 = j, and set$x_2 \equiv x_{2j}$. If no valid extension is found, backtrack to k=1 and resume from j= j_1+1: $x^1 = [x_{1j_1+1}]$.
- …
- Step k = n: Construct the full candidate solution $x^n = [x_1 \; x_2 \ldots x_{n-1} \; x_{nj}]^T$. For each j=1…$s_n$, check the continuity condition. If the condition is satisfied $c(x^n) = 1$, a valid solution $x^* = x^n$ is found. Otherwise, backtrack to $k = n-1$ and continue the search.

If initial values are predefined, they are included in the solution vector x*x* throughout all iterations. These fixed components guide the traversal and reduce the search space. For example, if $\exists i, \quad x^*(i) = x_i^0, \quad x_i^0 \in S_i$ then $x^1 = \left[x_i^0 \;\; x_{1j}\right]^T, x^2 = \left[x_i^0 \; x_1 \;\; x_{2j}\right]^T, \ldots, x^n = \left[x_i^0 \;\; x_1 \;\; \ldots \;\; x_{nj}\right]^T$.

4 Proposed Improvement Approach

The proposed improvement approach aims to reduce the number of iterations required to construct a valid solution vector x^* in combinatorial optimization problems. Unlike classical backtracking, which explores individual elements of the solution space S_i, this approach introduces meta-sets S^{kl} composed of structured vectors $x^{k,l}$ that satisfy sub-functions of the continuity criterion a priori:

$$c_P(x^{k,l}) = 1 \tag{2}$$

where the mentioned subfunction is denoted by c_p, and the expression of $x^{k,l}$ is

$$x^{k,l} = [x_k \quad \ldots \quad x_{l-1} \quad x_l]^T \tag{3}$$

Unlike the classical approach, which iterates over individual elements from the sets S_i, the proposed approach performs iterations over grouped sets S_{kl}, referred to as meta-sets: $S^{k,l} = S_k \times S_{k+1}, \ldots, S_l$, $S = \prod S^{k,l}$. These are formed by regrouping the original sets S_i, and the vectors $x^{k,l}$ are selected based on their compliance with the sub-function c_p. The final solution is obtained as follows by identifying combinations of these vectors that satisfy the full continuity criterion:

$$x^* = [x_{kl}^*]^T \tag{4}$$

This change in strategy transforms the search from an element-wise to a vector-wise process, which significantly reduces the number of iterations. Additionally, the approach introduces an ordered search mechanism: instead of traversing the sets in the natural sequence i =1, 2, … , *n*, the algorithm prioritizes meta-sets based on the cardinality of the vectors x^{kl}. This means that the search begins with the most constrained sets, those with the fewest valid combinations, thus minimizing the likelihood of backtracking.

This improvement approach is particularly effective in problems with structured constraints, such as Sudoku and Kakuro, described as follows, where the solution space can be narrowed significantly by applying continuity criteria early and grouping candidate solutions into meaningful subsets.

4.1 Tailoring the definition to the Sudoku problem

The Sudoku puzzle is the first illustration of the proposed improvement approach. The problem is defined on a grid where each row, column, and subgrid must contain distinct

digits from a predefined set (typically 1 to 9). The continuity criterion in this context ensures that no digit is repeated within any of these structures:

$$g(x) = \begin{cases} 1 & if \quad \wedge (g_L(x) = 1, \quad g_C(x) = 1, \quad g_Q(x) = 1) \\ 0 & otherwise \end{cases} \tag{5}$$

where

$$x = [x_{11} \quad \dots \quad x_{98} \quad x_{99}]^T, \quad x_{ij} \in S_{ij} = \{1,2,\dots,9\} \tag{6}$$

The indices ij refer, in this case, to row i and column j of the matrix associated with the Sudoku problem (see Figure 2 left):

$$g_L(x,i) = \begin{cases} 1 & if \quad \exists_1 x_{ij} = 1,\dots,9 \quad \forall j = 1,\dots,9 \quad i = 1,\dots,9 \\ 0 & otherwise \end{cases} \tag{7}$$

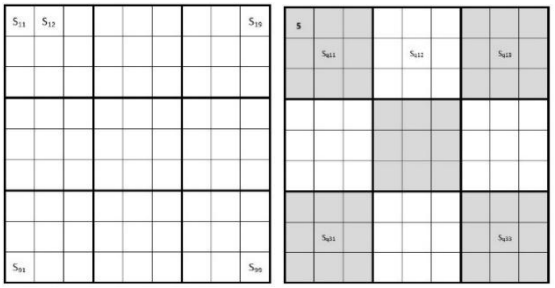


Figure 2

Illustration of the structure of the sets S_{ij} and S_{qij}, and how initial values are fixed in the solution

No matter what the row, the digits do not repeat, i.e.

$$g_C(x,j) = \begin{cases} 1 & if \quad \exists_1 x_{ij} = 1,\dots,9 \quad \forall i = 1,\dots,9 \quad j = 1,\dots,9 \\ 0 & otherwise \end{cases} \tag{8}$$

No matter the column, the digits do not repeat, i.e.

$$g_Q(x,i,j) = \begin{cases} 1 & if \quad \exists_1 x_{ij} = 1,\dots,9 \quad \forall x_{i,j} \in \vee \left(S_{q11},\dots S_{q33}\right) \\ 0 & otherwise \end{cases} \tag{9}$$

No matter the sub grid, the digits do not repeat, as illustrated in Figure 2 right:

$$S_{q11} = \cap (S_{11}, S_{12}, \dots, S_{33}) \tag{10}$$

In classical backtracking, the algorithm iteratively substitutes unknown elements (marked as 0) with digits in the set {1,2,…,9}, verifying the continuity condition at each step. The search graph is constructed by traversing the matrix row by row and column by column, applying the continuity criterion to determine whether a partial solution can be extended.

The Sudoku problem includes the specification of initial values, which are not modified during the search process and therefore also appear in the final solution. For example, $x^*(11) = x_{11}^0 = 5$.

4.2 Tailoring the definition to the Kakuro problem

For a structure composed of row and column vectors, the task is to determine the elements of these vectors such that their sum equals the predefined values K_i. An additional constraint requires that the vectors contain distinct elements (Figure 3):

$$S_i = \{x_i | x_i = [x_{i,1} \quad \dots \quad x_{i,ni}], x_{i,j} = 1, \dots, 9\}, i = 1, \dots, m \tag{11}$$

where

$$S = S_1 \times S_2 \times, \dots, S_m, x \in S \tag{12}$$

The objective function evaluates whether a given vector x_i satisfies the condition that all its elements are distinct and their sum equals the corresponding value K_i. If this condition is met for all vectors, the function returns 1; otherwise, it returns 0:

$$g(x) = \begin{cases} 1 \quad if \quad \left(\exists_1 x_{ij} = 1, \dots, 9; \forall j = 1, \dots, ni; i = 1, \dots, m\right) \quad \wedge \quad \sum_{j=1}^{ni} x_{ij} = K_i \\ 0 \quad otherwise \end{cases} \tag{13}$$

Some vectors may share common elements, i.e., there exist indices u and v such that a non-empty intersection is obtained, namely $\exists u, vx\ x_u \cap x_v \neq \Phi$. This implies that certain vectors have overlapping values.

The proposed solution for the Kakuro puzzle also relies on narrowing the domain of possible solutions. An initial brute-force approach might assume that the unknown elements of the vectors can take any natural value from 1 to 9. However, this method refines that assumption by introducing several criteria aimed at reducing the number of feasible solutions:

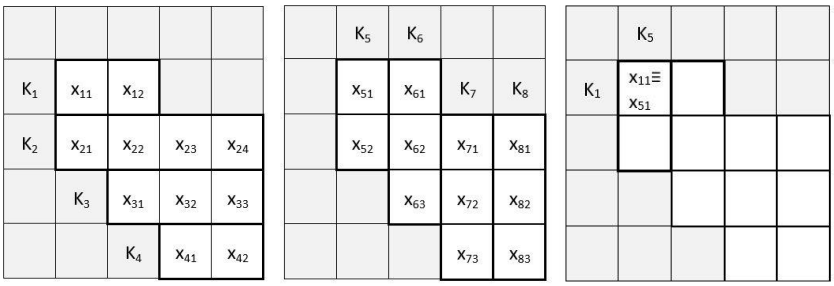


Figure 3
Left and center: example illustrating the definition of vectors $x_1,\dots,x_8$,
right: highlighting that vectors x_1 and x_5 share a common element

- Criterion 1: Each vector x_i contains nine distinct elements. This constraint limits the solution space to permutations of 9 elements taken n_i at a time:

$$S_i = \left\{[x_{i,1}, \dots, x_{i,ni}] | \exists_1 x_{i,j} = 1, \dots, 9; \forall j\right\}, |S_i| = P(9,\text{ni}) = \frac{9!}{(9-\text{ni})!} \tag{14}$$

- Criterion 2: The sum of the elements in vector x_i must equal a predefined value K_i. This condition further reduces the domain S_i by eliminating all vectors that do not satisfy the required sum:

$$S_i = \{[x_{i,1}, \dots, x_{i,ni}] | \wedge (\exists_1 x_{i,j} = 1, \dots, 9; \forall j, \sum_{j=1}^{ni} x_{i,j} = K_i)\} \quad (15)$$

The action of the first two criteria is illustrated in Figure 4.

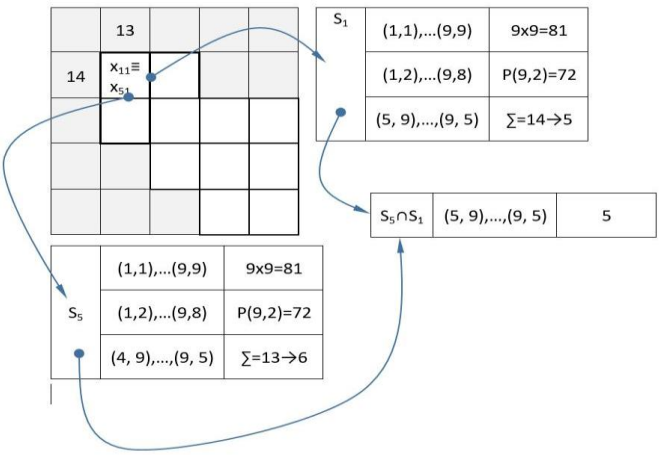


Figure 4
Reduction of possible variants through the application of the first two criteria

- Criterion 3: Vectors share common elements, which leads to a reduction in the number of valid combinations:

$$if\ x_u \cap x_v = x_{u,uj} = x_{v,vj} \Rightarrow$$
$$S_u = \{[x_{u,1}, \dots, x_{u,uj}, \dots, x_{u,nu}] | \wedge (\exists_1 x_{u,j} = 1, \dots, 9; \forall j, \sum_{j=1}^{nu} x_{u,j} = K_u)\} \wedge \quad (15)$$
$$S_v = \{[x_{v,1}, \dots, x_{v,vj}, \dots, x_{v,nv}] | \wedge (\exists_1 x_{v,j} = 1, \dots, 9; \forall j, \sum_{j=1}^{nv} x_{v,j} = K_v)\}$$

Criterion 3 can be applied iteratively in two ways:

- Row–column: where all columns are analyzed successively based on the rows.
- Column–row: where all rows are analyzed successively based on the columns.

To ensure a continuous reduction in the solution space, these iterations are grouped into pairs: row–column and column–row.

5 Case Studies

To validate the effectiveness of the proposed improvement approach, the Sudoku and Kakuro problems are exemplified in this section in terms of two case studies. Each case study includes a comparison between the classical backtracking algorithm and

the proposed approach applied as an algorithm, highlighting differences in iteration count, convergence behavior, and domain reduction. Through these case studies, it is proved how the proposed approach can be adapted to different problem structures and how it performs under both moderate and high-difficulty scenarios.

5.1 Sudoku case study

A simplified 3x3 Sudoku puzzle is first treated. This example demonstrates how the proposed approach reduces the number of iterations compared to the classical backtracking. Figure 5 illustrates this problem.

In the classical approach, the solution space is defined as

$$S = S_{11} \times S_{12} \times, \dots, S_{33}, \text{ where each } S_{ij} = \{1,2,3\} \tag{16}$$

Some values are predefined and fixed throughout the search:

$$x^*(11) = x_{11}^0 = 1, x^*(13) = x_{13}^0 = 3, x^*(33) = x_{33}^0 = 1 \tag{17}$$

Unknown elements are marked with 0, which do not belong to the sets S_{ij}. The continuity condition is defined as

$$c(x,i,j) = \wedge \left(c_L(x,i), c_C(x,j)\right) \tag{18}$$

where

$$\begin{aligned} c_L(x,i) &= \begin{cases} 1 & if \quad \exists_1 x_{ij} = 1,\dots,3 \quad \forall j = 1,\dots,3 \quad i = 1,\dots,3 \\ 0 & otherwise \end{cases} \\ c_C(x,j) &= \begin{cases} 1 & if \quad \exists_1 x_{ij} = 1,\dots,3 \quad \forall i = 1,\dots,3 \quad j = 1,\dots,3 \\ 0 & otherwise \end{cases} \end{aligned} \tag{19}$$

1	0	3
0	0	0
0	0	1

1	S_{12}	2
S_{21}	1	S_{23}
S_{31}	S_{32}	1

1	2	3
3	1	2
2	3	1

Figure 5

Proposed 3x3 Sudoku problem and its solution

The algorithm constructs candidate vectors and verifies the continuity condition at each step. If the condition fails, the algorithm backtracks to a previous state. This behavior is illustrated in the search graph and the step-by-step execution shown in Figure 6 and Table 1.

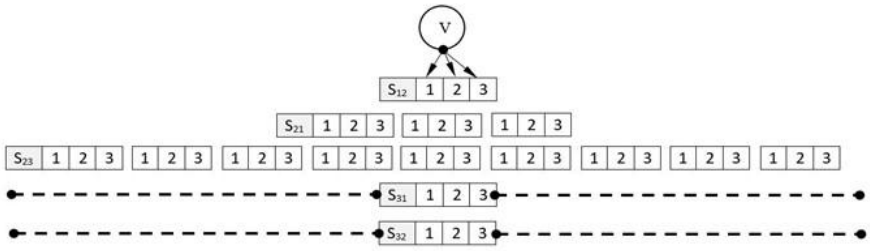


Figure 6

Search graph and backtracking steps

Table 1

Classical algorithm steps, including values of k, proposed vectors, and continuity checks

Nr	k	Proposed vector	Continuity criteria	
1	12	x=[1,1,3;0,0,0;0,0,1]	No	$c_L(x)=0$
2	12	x=[1,2,3;0,0,0;0,0,1]	Yes	
3	21	x=[1,2,3;1,0,0;0,0,1]	No	$c_C(x)=0$
4	21	x=[1,2,3;2,0,0;0,0,1]	Yes	
5	22	x=[1,2,3;2,1,0;0,0,1]	Yes	
6	23	x=[1,2,3;2,1,1;0,0,1]	No	$c_L(x)=0$
7	23	x=[1,2,3;2,1,2;0,0,1]	No	$c_L(x)=0$
8	23	x=[1,2,3;2,1,3;0,0,1]	No	$c_C(x)=0$
9	22	x=[1,2,3;2,2,0;0,0,1]	No	$c_L(x)=0$, $c_C(x)=0$
10	22	x=[1,2,3;2,3,0;0,0,1]	Yes	
11	23	x=[1,2,3;2,3,1;0,0,1]	No	$c_C(x)=0$
12	23	x=[1,2,3;2,3,2;0,0,1]	No	$c_L(x)=0$
13	23	x=[1,2,3;2,3,3;0,0,1]	No	$c_L(x)=0$
14	21	x=[1,2,3;3,0,0;0,0,1]	Yes	
15	22	x=[1,2,3;3,1,0;0,0,1]	Yes	
16	23	x=[1,2,3;3,1,1;0,0,1]	No	$c_L(x)=0$, $c_C(x)=0$
17	23	x=[1,2,3;3,1,2;0,0,1]	Yes	
18	31	x=[1,2,3;3,1,2;1,0,1]	No	$c_L(x)=0$, $c_C(x)=0$
19	31	x=[1,2,3;3,1,2;2,0,1]	Yes	
20	32	x=[1,2,3;3,1,2;2,1,1]	No	$c_L(x)=0$
21	32	x=[1,2,3;3,1,2;2,2,1]	No	$c_L(x)=0$
22	32	x=[1,2,3;3,1,2;2,3,1]	Yes	$g(x)=0$

Backtracking is observed at steps such as rows 9 and 14, where the algorithm returns from $k = 23$ to $k = 22$, and from $k = 23$ to $k = 21$, respectively. In this example, the classical approach explores up to 59,049 (3^{10}) possible substitutions.

The proposed approach improves this further by constructing meta-sets of valid row vectors that satisfy the row continuity condition c_L. These vectors are selected such that each digit appears only once per row, and the digit 0 (used to mark unknowns) is excluded from the solution space:

$$\begin{aligned} S^1 &= \{[1,2,3]\} \\ S^2 &= \{[1,2,3],[1,3,2],\dots,[3,1,2]\} \\ S^3 &= \{[2,3,1],[3,2,1]\} \end{aligned} \tag{20}$$

These vectors are then combined and filtered using the column continuity criterion c_C, which ensures that each digit appears only once per column. This two-step filtering process significantly reduces the number of valid combinations as shown in Figure 7 and Table 2.

The algorithm could begin by enforcing the row constraint c_L and then verifying the column constraint c_C. In the case of Sudoku, this strategy can be implemented directly by transposing the initial matrix.

This application demonstrates how the proposed approach leverages structural constraints and search space ordering to reduce computational effort. By starting with the most constrained sets and applying continuity filters early, the algorithm avoids unnecessary backtracking and converges efficiently to a valid solution.

Following the first elementary example, the performance of the proposed approach is examined on a more complex 9x9 Sudoku puzzle. This case study includes both a medium-difficulty (Figure 8) and two high-difficulty scenarios to demonstrate the scalability of the optimization strategy.

In the medium-difficulty scenario (Figure 8), the classical approach required 175 substitution attempts, while the proposed approach identified a valid solution using only one variant per row (Figure 9). This demonstrates the efficiency of the meta-set strategy in reducing the search space and avoiding redundant computations. The example has a medium level of difficulty (it can be solved deductively) and confirms that the proposed approach not only accelerates the search process but also maintains solution accuracy, even as the problem size increases.

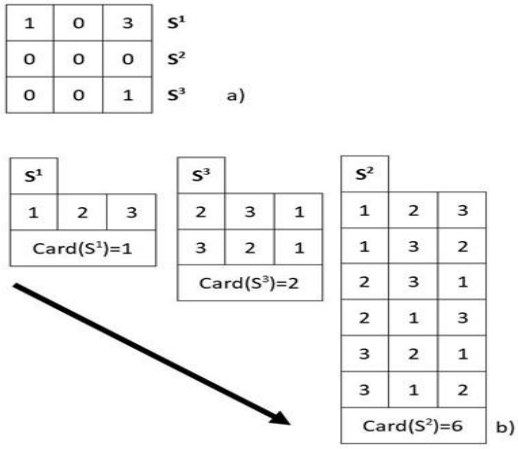


Figure 7

Assigning the meta-sets a), and defining the variables b)

Table 2

The steps in solving the problem illustrated in Figure 7

Nr	k	Proposed vector	Continuity criteria	
1	1	x=[[1,2,3];0,0,0;0,0,1]	Yes	c_C (x)=1

2	3	x=[[1,2,3];0,0,0;[2,3,1]]	Yes	$c_C(x)=1$
3	2	x=[[1,2,3];[1,2,3][2,3,1]]	No	$c_C(x)=0$
4	2	x=[[1,2,3];[1,3,2][2,3,1]]	No	$c_C(x)=0$
…	2	…	No	$c_C(x)=0$
8	2	x=[[1,2,3];[3,1,2][2,3,1]]	Yes	$c_C(x)=1$

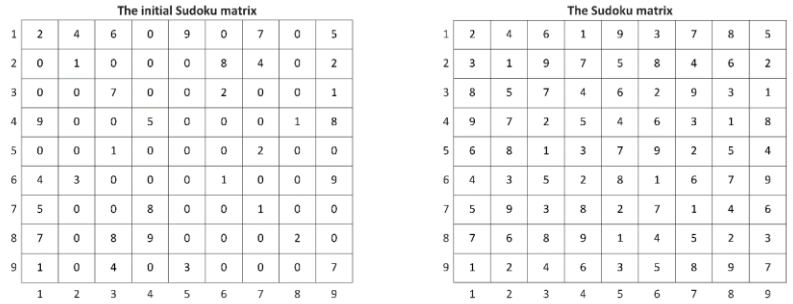


Figure 8

Initial configuration and final solution of a medium-level Sudoku game

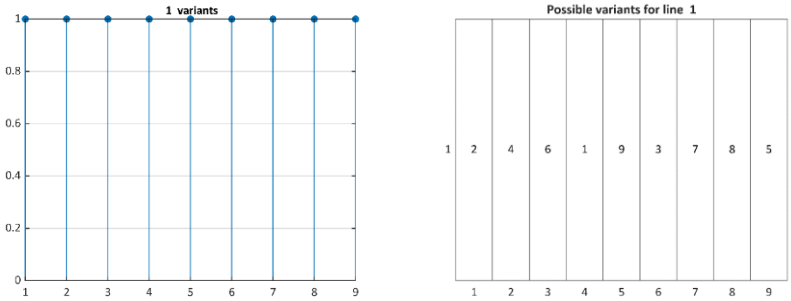


Figure 9

Analysis of the search algorithm left: number of variants for each row,
right: possible variants for row 1

The two high-difficulty Sudoku puzzles and scenarios are designed to highlight how changes in the strategy for defining variant sets, whether by rows or by columns, can significantly affect the number of iterations required. The two puzzles presented in Figure10 and Figure 11 differ only in the orientation of their initial matrices: one is the transpose of the other. This setup allows us to observe how the choice of traversal direction influences the efficiency of the algorithm.

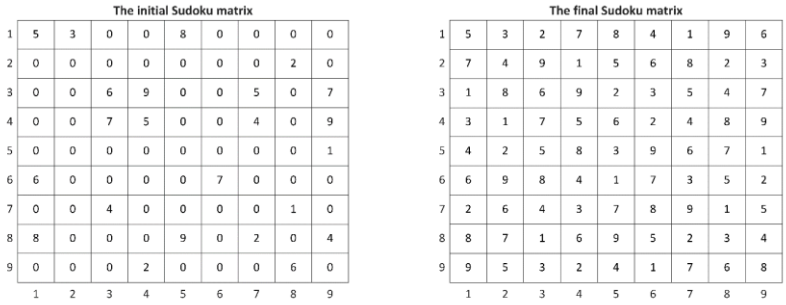


Figure 10

Initial configuration and final solution of a high-level Sudoku game (scenario 1)

In the first scenario, the classical approach required 244,845 iterations, while the proposed approach converged in 56,410 trials. Each row was analyzed to generate valid variants, and the final selections are marked in the solution matrix shown in Figure 12.

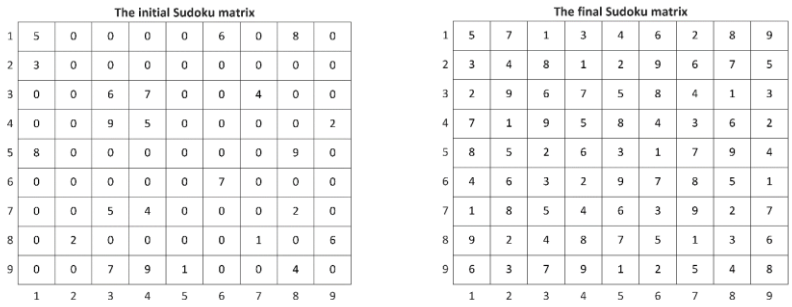


Figure 11

Initial configuration and final solution of a high-level Sudoku game (scenario 2)

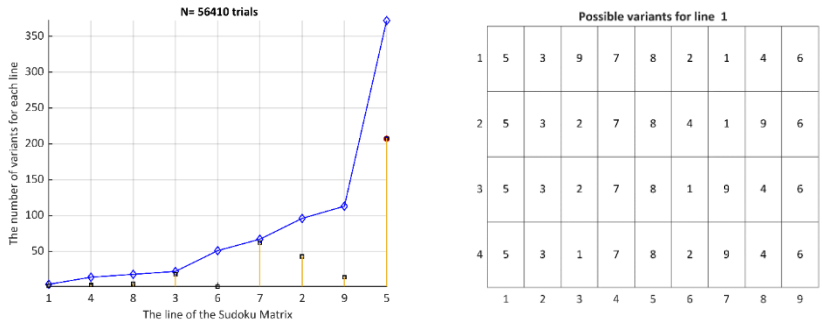
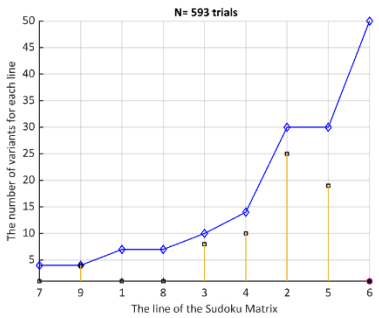


Figure 12

Analysis of the search algorithm in the scenario 1: left: number of variants for each line, right: possible variants for row 1

In scenario 2 (Figure 11), the classical approach needed 26,107 iterations, whereas the proposed approach solved the puzzle using only 593 trials. This dramatic reduction demonstrates the impact of choosing an optimal direction for variant generation and traversal (Figure 13).

These case studies confirm that the proposed method is not only scalable but also adaptable to different structural configurations. By leveraging meta-set construction and continuity filtering, the algorithm maintains high performance even in scenarios where deductive solving is impractical.



Possible variants for line 1

1	5	7	1	3	4	6	2	8	9
2	5	7	1	2	4	6	9	8	3
3	5	7	1	2	4	6	3	8	9
4	5	4	1	3	2	6	9	8	7
5	5	4	1	3	2	6	7	8	9
6	5	4	1	2	3	6	9	8	7
7	5	4	1	2	3	6	7	8	9
	1	2	3	4	5	6	7	8	9

Figure 13

Analysis of the search algorithm in the scenario 2: left: number of variants for each line, right: possible variants for row 1

5.2 Kakuro case study

The problem is defined in Figure 14 its solution is sketched in Figure 15. The simulation of the approach separated the vectors x_i into two classes: rows and columns. The sets of possible variants S_i were generated based on the first two criteria; by permuting digits and selecting only those permutations whose sum equals K_i.

		C1	C2	C3/4	C5/6	C7	C8
				16↓	23↓		
L1		30↓	17→ 29↓			11↓	10↓
L2	35→						
L3/4	13→			16→ 24↓			
L5/6	24→				6→ 17↓		
L7	33→						
L8			16→				

Figure 14

Proposed puzzle: there are 16 vectors (8 Rows and 8 Columns). The sum values $K_1,\ldots,K_{16}$ are shown along with arrows indicating the corresponding vectors

The initial complexity, highlighted in Figure 16, shows that the initial number of variants for the rows, 10^{12}, is still inadequate for a brute-force approach. To address this, the third criterion was applied: eliminating those variants that do not allow common values at the intersections of rows and columns. This filtering was performed iteratively. Six successive pairs of iterations, illustrated in Figure 16 and Figure 17, ultimately led to a single valid variant, which corresponds to the solution to the problem.

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	9	8	0	0
3	0	9	8	7	6	2	3
4	0	8	5	0	9	3	4
5	0	7	9	8	0	5	1
6	0	6	7	9	8	1	2
7	0	0	0	7	9	0	0

Figure 15

Final solution: Obtained using 6 pairs of iterations (row–column type)

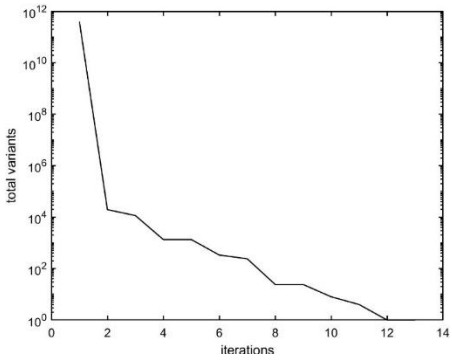


Figure 16

Change in the size of domain *S* after successive iterations. *S* was calculated for the entire set of rows

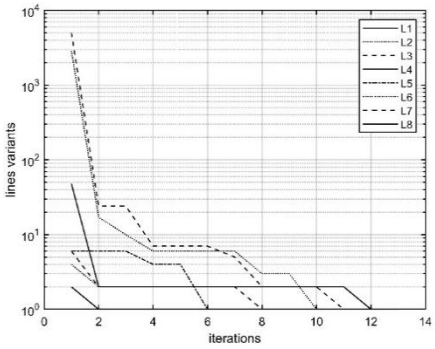


Figure 17
Change in the size of domain S_i for each individual row after successive iterations

This convergence to the solution effectively avoided the need for brute-force methods. This also demonstrates the efficiency of the proposed approach in handling high-complexity combinatorial puzzles.

For a more illustrative visualization, the representations were made using a semi-logarithmic scale.

5.3 Comparative analysis

To assess the performance of the proposed improvement approach, case studies were conducted using Sudoku and Kakuro puzzles of varying complexity. In the case of the 3x3 Sudoku puzzle, the classical approach explored 59,049 possible substitutions, while the improvement approach reached a solution using only 22 iterations. This reduction was achieved by grouping elements into meta-sets and applying continuity criteria early in the search process. Similarly, for the same puzzle, the improvement approach used only 8 out of 12 possible substitutions, compared to the exhaustive approach of the classical algorithm.

For a medium difficulty 9x9 Sudoku puzzle, the classical approach required 175 substitution attempts, whereas the proposed approach identified a valid solution with just one attempt per row. This demonstrates the efficiency of the meta-set strategy in reducing the search space and avoiding redundant computations. In high-difficulty scenarios, the classical approach needed up to 244,845 iterations, while the improvement approach converged in 56,410 iterations, highlighting a substantial improvement.

The Kakuro puzzle further illustrated the benefits of the proposed approach. By applying three filtering criteria, distinct elements, predefined sum constraints, and shared values across vectors, the solution space was significantly reduced. Iterative refinement through row-column and column-row pairings led to convergence after only six iteration pairs, avoiding brute-force enumeration. The final solution was achieved with a minimal number of trials, and the evolution of the domain size was visualized using a semi-logarithmic scale to emphasize the efficiency gains.

These case studies can be easily deployed, after appropriate adjustment if different non-combinatorial optimization problems are solved, in other real-world applications. Such applications could be selected in several fields including evolving controllers [21], Internet-of-Things in robotics and navigation [22] [23], electric vehicles and ad-hoc networks [24-26], haptic interfaces [27], Kalman filters [28], fuzzy control [29-32], learning control [33] [34], flexible structures [35], human well-being and health [36] [37].

Conclusions

This paper has introduced a novel improvement approach to the classical backtracking algorithms, aimed at improving their efficiency in solving complex combinatorial problems. By restructuring the search space into meta-sets and applying continuity criteria

early in the traversal process, the proposed approach significantly reduces the number of iterations required to reach valid solutions.

Beyond its immediate applications, the proposed approach offers a scalable framework for tackling a wide range of constraint satisfaction and combinatorial optimization problems. Its adaptability to different problem structures and its compatibility with existing heuristic and pruning strategies make it a promising candidate for integration into hybrid algorithmic systems. Moreover, the reduction in computational overhead opens new possibilities for real-time or resource-constrained environments where classical backtracking would be impractical.

Future work will explore the extension of this approach to dynamic or probabilistic problem domains, as well as its integration with machine learning techniques for predictive pruning and meta-set generation. The insights gained from this study contribute to the ongoing evolution of backtracking algorithms, reaffirming their relevance and adaptability in modern computational contexts.

Acknowledgement

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI - UEFISCDI, project number ERANET-ENUAC-e-MATS, within PNCDI IV, and the Széchenyi István University, Győr, Hungary.

References

[1] D. E. Knuth: The Art of Computer Programming, Volume 4: Combinatorial Algorithms, Part 1, Addison-Wesley, 2011

[2] T. H. Cormen et al.: Introduction to Algorithms, MIT Press, 2009

[3] J. Kleinberg, É. Tardos: Algorithm Design, Pearson, 2005

[4] M. A. Weiss: Data Structures and Algorithm Analysis in C++, Pearson, 2013

[5] R. Sedgewick, K. Wayne: Algorithms, Addison-Wesley, 2011

[6] C. H. Papadimitriou, K. Steiglitz: Combinatorial Optimization: Algorithms and Complexity, Dover Publications, 1998

[7] C. H. Papadimitriou: Computational Complexity, Addison-Wesley, 1994

[8] A. V. Aho et al.: The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974

[9] T. H. Cormen: Algorithms Unlocked, MIT Press, 2013

[10] D. Harel, Y. Feldman: Algorithmics: The Spirit of Computing, Addison-Wesley, 2004

[11] S. S. Skiena: The Algorithm Design Manual, Springer, 2008

[12] R. Sedgewick: Algorithms in C++, Addison-Wesley, 1998

[13] R. Lafore: Data Structures and Algorithms in Java, Sams Publishing, 2002

[14] A. Levitin: Introduction to the Design and Analysis of Algorithms, Pearson, 2011

[15] M. T. Goodrich, R. Tamassia: Algorithm Design and Applications, Wiley, 2014

[16] T. Roughgarden: Algorithms Illuminated (Part 3), Greedy Algorithms and Dynamic Programming, 2019

[17] K. Mehlhorn, P. Sanders: Algorithms and Data Structures: The Basic Toolbox, Springer, 2008

[18] A. Levitin, M. Levitin: Algorithmic Puzzles, Oxford University Press, 2011

[19] A. Aziz, T.-H. Lee: Algorithms for Interviews, EPI, 2010

[20] J. Bentley: Programming Pearls, Addison-Wesley, 2000

[21] B. Costa, I. Škrjanc, S. Blažič, P. Angelov: A practical implementation of self-evolving cloud-based control of a pilot plant, in Proceedings of 2013 IEEE International Conference on Cybernetics, Lausanne, Switzerland, 2013, pp. 7-12

[22] J. Hvizdoš, I. Vojtko, M. Koscelanský, J. Pavlov, J. Vaščák, P. Sinčák: Applications of remote controlled robotics in the intelligent space, in Proceedings of IEEE 15th International Symposium on Applied Machine Intelligence and Informatics, Herl'any, Slovakia, 2017, pp. 117-122

[23] C. Pozna, R.-E. Precup, P. Földesi: A novel pose estimation algorithm for robotic navigation, Robotics and Autonomous Systems, Vol. 63, 2014, pp. 10-21

[24] S. I. Boucetta, Z. C. Johanyák: Optimized ad-hoc multi-hop broadcast protocol for emergency message dissemination in vehicular ad-hoc networks, Acta Polytechnica Hungarica, Vol. 19, No. 5, 2022, pp. 23-42

[25] A. Lucchini, S. Formentin, M. Corno, D. Piga, S. M. Savaresi: Torque vectoring for high-performance electric vehicles: a data-driven MPC approach, IEEE Control Systems Letters, Vol. 4, No. 3, 2020, pp. 725-730

[26] R.-E. Precup, C.-A. Bojan-Dragos, K. Gao, S. Cui: Modeling results of trajectory planning in an intersection crossing scenario with a connected autonomous electric bus and multiple human driven vehicles, in Proceedings of the 2025 KES International Conference on Smart Transportation Systems, Solin, Croatia, 2025, pp. 1-13

[27] N. Ando, P. Korondi, H. Hashimoto: Networked telemicromanipulation systems "Haptic Loupe", IEEE Transactions on Industrial Electronics, Vol. 51, No. 6, 2004, pp. 1259-1271

[28] J. Kuti, I. J. Rudas, H.-J. Gao, P. Galambos: Computationally relaxed unscented Kalman filter, IEEE Transactions on Cybernetics, Vol. 53, No. 3, 2023, pp. 1557-1565

[29] R.-E. Precup, S. Preitl: Popov-type stability analysis method for fuzzy control systems, in Proceedings of Fifth European Congress on Intelligent Technologies and Soft Computing, Aachen, Germany, 1997, Vol. 2, pp. 1306-1310

[30] R. E. Haber, J. R. Alique, A. Alique, J. Hernández, R. Uribe-Etxebarria: Embedded fuzzy-control system for machining processes: Results of a case study, Computers in Industry, Vol. 50, No. 3, 2003, pp. 353-366

[31] R.-E. Precup, S. Preitl, M. Balas, V. Balas: Fuzzy controllers for tire slip control in anti-lock braking systems, in Proceedings of 2004 IEEE International Conference on Fuzzy Systems, Budapest, Hungary, 2004, vol. 3, pp. 1317-1322

[32] R.-E. Precup, T. Haidegger, S. Preitl, B. Benyó, A. S. Paul, L. Kovács: Fuzzy control solution for telesurgical applications, Applied and Computational Mathematics, Vol. 11, No. 3, 2012, pp. 378-397

[33] G. Paczolay, I. Harmati: NPV-DQN: Improving value-based reinforcement learning, by variable discount factor, with control applications, Acta Polytechnica Hungarica, Vol. 21, No. 11, 2024, pp. 175-190

[34] I. A. Zamfirache, R.-E. Precup, E. M. Petriu: Adaptive reinforcement learning-based control using proximal policy optimization and slime mould algorithm with experimental tower crane system validation, Applied Soft Computing, Vol. 160, 2024, paper 111687

[35] P. Milić, D. Marinković, S. Klinge, Ž. Ćojbašić: Reissner-Mindlin based isogeometric finite element formulation for piezoelectric active laminated shells, Tehnički Vjesnik, Vol. 30, No. 2, 2023, pp. 416-425

[36] F. G. Filip: Automation and computers and their contribution to human well-being and resilience, Studies in Informatics and Control, Vol. 30, No. 4, 2021, pp. 5-18

[37] B. Gergics, M. Puskás, L. Kisbenedek, M. Dömény, L. Kovács, D. A. Drexler: Chemotherapy optimization and patient model parameter estimation based on noisy measurements, Acta Polytechnica Hungarica, Vol. 21, No. 10, 2024, pp. 475-494