On an Improvement to the Next-step Procedure

Nikita Kosyanov [1]

[1] *Institute for System Dynamics and Control Theory of SB RAS, Lermontov str., 134, Irkutsk, 664033, Russia*

Abstract

In this paper, we address a next-step procedure for solving nonsmooth integer programming problems, where the objective function is given as a maximum and the constraints include both equalities and inequalities. We propose two modifications of the original procedure aimed at significantly reducing the number of operations required to obtain a solution.

Keywords

Integer programming, nonsmooth optimization, minimax problem, next-step procedure

1. Introduction

Modern computing is inseparable from parallel processing. It is now virtually impossible to find a computer or smartphone equipped with a single-core processor. Moreover, many devices are equipped with heterogeneous cores—high-performance and energy-efficient—requiring specially designed software to balance the computational load across them.

Load balancing problems of this kind are relevant not only for large-scale computing centers [1], where workloads must be distributed across multiple compute clusters, but also for service systems such as healthcare, where, for example, patients in infectious disease hospitals must be allocated among available physicians [2].

In many cases, such multi-resource systems can be described mathematically, and the task of assigning jobs or requests can be formulated as an optimization problem—typically, one that minimizes the maximum load across all resources. However, these problems often include the constraint that individual tasks (or patients) cannot be divided among multiple resources, which necessitates the development and application of specialized solution methods that explicitly account for such indivisibilities.

2. Next-step procedure

We consider the problem $(\mathcal{TP}_N)$ of performance optimization in the case of parallel processing of N computational scenarios using n processing resources:

$$(\mathcal{TP}_N): f_N(x) = \max\left\{\frac{x_1}{a_1}; \frac{x_2}{a_2}; \dots; \frac{x_n}{a_n}\right\} \downarrow \min_{x \in X_N}, \quad X_N \subset \mathbb{N}_0^n, \tag{1}$$

$$X_N: \sum_{i=1}^{n} x_i = N, \quad 0 \le x_i \le c_i, \quad i = \overline{1,n}. \tag{2}$$

where x_i is the number of scenarios processed by resource i; $a_i \in \mathbb{N}$ denotes the performance (capacity) of resource i; X_N is the available set of values for x, which is a subset of the n-dimensional vector space with non-negative integer coordinates ($\mathbb{N}_0^n$) and satisfies the constraints (3.2); $N \ge 0$ represents the total number of scenarios to be processed; $c_i \ge 0$ is the maximum number of scenarios (in terms of execution time or memory usage) that resource i can handle.

7th International Workshop on Information, Computation, and Control Systems for Distributed Environments (ICCS-DE 2025), July 7–11, 2025, Irkutsk, Russia

EMAIL: kosyanov.nik@gmail.com (A. 1);

ORCID: 0009-0005-3448-5573 (A. 1);

ICCS-DE 2025 Workshop Proceedings

DOI: 10.47350/ICCS-DE.2025.20

In addition, we define the function $p(x)$:

$$p(x) \triangleq (p_1, p_2, \dots, p_n) = \left(\frac{x_1+1}{a_1}, \frac{x_2+1}{a_2}, \dots, \frac{x_n+1}{a_n}\right), \quad x \in \mathbb{N}_0^n. \tag{3}$$

We are now in a position to describe the solution procedure for problem $(\mathcal{TP}_N)$.

Next-step Procedure (NSP)

Step 0. Set $k := 0$, $x^k := 0$, $p(x^k) := \left(\frac{1}{a_1}, \frac{1}{a_2}, \dots, \frac{1}{a_n}\right)$.

Step 1. IF $k < N$, THEN find $j = \underset{i\in\{1,2,\dots,n\}}{\operatorname{argmin}} \{p_i\}$, where $x_j + 1 \le c_j$, ELSE STOP.

Step 2. Set

$$x^{k+1} := \begin{cases} x_i^{k+1} = x_i^k, & \forall i \neq j, \\ x_j^k = x_j^k + 1; \end{cases} \quad p(x^{k+1}) := \begin{cases} p_i^{k+1} = p_i^k, & \forall i \neq j, \\ p_j^{k+1} = p_j^k + \dfrac{1}{a_j}; \end{cases} . \tag{4}$$

$k := k + 1$ and loop to Step 1.

2.1. First improvement

It is evident that applying the Next-step procedure to solve the problem requires $o(N)$ operations. In practical cases, the value of N is often significantly large (e.g., exceeding 100,000), which inevitably leads to a considerable slowdown in code execution. Nevertheless, it is not required to begin the search with $k = 0$; rather, it is enough to identify an initial vector x^{k_1} with a uniform distribution of the k scenarios. For example, let $N_0 = N$ and

$$x^{k_1}: \begin{cases} x_i^{k_1} = q_1 a_i, & if\ q_1 a_i, < c_i, \\ x_i^{k_1} = c_i, & if\ q_1 a_i, \ge c_i; \end{cases} \quad q_1 = \left\lfloor \frac{N_0}{\sum_{i=1}^n a_i} \right\rfloor, \quad k_1 = \sum_{i=1}^n x_i^{k_1}. \tag{5}$$

where $\lfloor \dots \rfloor$ denotes the floor function, which rounds a number down to the nearest integer.

If, for some $x_i^{k_1}$, we have $x_i^{k_1} = c_i$, this means that the inequality corresponding to this component holds as an equality, and the associated resource processes the maximum number of scenarios it can handle. Hence, the number of unprocessed scenarios is calculated as:

$$N_1 = N_0 - k_1. \tag{6}$$

We can repeat this allocation process (excluding fully loaded resources), while $N_s < N_{s+1}$, where s is a counter. Thus, the computational cost of the NSP is equal to $o(n) + o(N_s)$.

2.2. Second improvement

The second improvement to the Next-Step procedure concerns the vector function $p(x)$. It can be observed that at each iteration, only a single component of $p(x)$ changes, while the others remain unchanged. Thus, if we sort the components of $p(x)$ in ascending order $\tilde{p}(x) = (p(x), \le)$, the desired component j (Step 1 in NSP) will appear in the first position.

Then, after each step, there is no need to search for the minimum in the entire array (which has a computational complexity of $o(n)$); it is sufficient to compare the updated first component with the second one (with computational complexity $o(1)$) and, if necessary, adjust its position relative to the others (with complexity $o(n)$). This implementation can lead to a significant reduction in the number of operations in certain cases.

3. Numerical testing

We conducted numerical testing on a set of test examples for the original procedure, as well as for its three modifications: with Improvement 1, with Improvement 2, and with both improvements combined. The testing results are presented in Table 1.

Table 1
NSP modification testing

Type	Average execution time (s)	Memory usage (GB)
Original NSP	4.2174 s	1.1824
NSP with Improvement 1	0.0114 s	1.1825
NSP with Improvement 2	3.9546 s	1.1824
NSP with both Improvements	0.0107	1.1825

4. Conclusions

The numerical results demonstrate that the first improvement to the NSP significantly accelerates code execution—by a factor of 371. In contrast, the second improvement alone yields only a modest speedup of 1.06 times. However, when both improvements are applied simultaneously, the overall performance increases dramatically, resulting in a 394-fold reduction in execution time. At the same time, the memory usage of the computing system increases only slightly.

5. Acknowledgements

The research was funded by the Ministry of Education and Science of the Russian Federation within the framework of the project "Theoretical foundations, methods and high-performance algorithms for continuous and discrete optimization to support interdisciplinary research" (No. of state registration: 121041300065-9, code FWEW-2021-0003).

6. References

[1] G. A. Oparin, A. G. Feoktistov, S. A. Gorsky, Management of asynchronous computational process in ORLANDO TOOLS, Vestnik kompyuternyh i informacionnyh tekhnologiy (2011) 48–55. doi:10.1145/1188913.1188915.

[2] N. Kosyanov, Epidemic optimal control problem with vaccination and isolation processes, in: Proceedings of the 8th International School-Seminar on Nonlinear Analysis and Extremal Problems (NLA-2024), ISDCT SB RAS, Irkutsk, Russia, 2024, pp. 137–139.