Page Faults Minimization for Virtual Memory Systems Using Working Set Strategy

Aslanbek Murzakhmetov * , Gaukhar Borankulova , Arseniy Bapanov , Zhanna Sadirmekova , Gabit Altybaev

Posted Date: 7 July 2025

doi: 10.20944/preprints202507.0494.v1

Keywords: page fault; page replacement algorithms; working set; program behavior; memory management

 doi:10.20944/preprints202507.0494.v1

Article

Page Faults Minimization for Virtual Memory Systems Using Working Set Strategy

Aslanbek Murzakhmetov [1,2,*], Gaukhar Borankulova [2], Arseniy Bapanov [2], Zhanna Sadirmekova [2] and Gabit Altybaev [3]

[1] School of Information Sciences, University of Illinois Urbana-Champaign, IL, 61820, USA
[2] Department of Information Systems, Faculty of Technology, M.Kh. Dulaty Taraz University, Taraz, 080001, Kazakhstan
[3] Department of Radio Engineering, Electronics and Telecommunications, International Information Technologies University, Almaty 050040, Kazakhstan
* Correspondence: aslanbek@illinois.edu

Abstract

Poor code locality in virtual memory systems is one of the reasons for page faults and, consequently, slow operation of an entire system. Despite the extensive body of research dedicated to minimizing page faults, the proposed solutions, which are predominantly based on clustering techniques, fail to provide approximation errors relative to an unknown optimal or near-optimal solution. We use Working Set strategy and geometric interpretation of the computational process, which clarifies the subtleties of optimization and facilitates the development of a mathematical model for minimization of page faults. Our approach includes functionals and constraints that define a set of possible solutions, which may be useful for future research aimed at developing an algorithm to achieve an optimal or ε-optimal solution. The results pave the way for researching and finding an efficient and cost-effective replacement algorithm similar to the working set approach.

Keywords: page fault; page replacement algorithms; working set; program behavior; memory management

1. Introduction

Virtual memory is a technique in memory management that allows a computer to use more memory than is physically available by temporarily transferring data from Random Access Memory (RAM) to disk storage. This is essential for running large programs or multiple applications simultaneously without running out of physical memory [1]. Virtual memory allows processes to use more memory than is physically available by swapping data between physical memory and secondary storage. This is a key mechanism in systems with virtual memory, where memory is divided into pages and the operating system dynamically manages their placement between RAM and disk [2,3]. However, this can lead to page faults. It happens when a process tries to access a virtual memory address for which there's currently no valid mapping in physical RAM and this is called "minor fault".

Page faults can also be the result of poor code locality. Poor code locality directly affects the frequency of page faults because it determines how often and in what order the program accesses memory pages [4–6]. Poor code locality causes the program to switch between pages frequently and can lead to thrashing (the system spends more time paging than executing code). The problem is how to relocate blocks (or program segments) across pages of virtual memory to minimize page faults [7,8]. Program code transformations, such as program restructuring [9–11] and refactoring [12–16], as well as various forms of code reorganization, have a positive impact on page faults, particularly in terms of locality. The absence of a definitive solution to this problem has sustained ongoing interest and research in both past and present studies.

 doi:10.20944/preprints202507.0494.v1

Many formulations of page faults or program optimization problems lead to complex combinatorial challenges, making it necessary in practice to rely on approximate or heuristic approaches. Most existing research in this area is based on clustering techniques [17,18]. While these techniques have shown improvements in experimental settings, they provide only approximate solutions with unknown accuracy, i.e., the cluster approach does not estimate how the solutions are obtained far or close to unknown exact (optimal) solutions [19,20]. The Working Set strategy, proposed by P.Denning [21], aims to prevent thrashing (excessive swapping that slows down the system) by ensuring that the pages a process needs are resident in memory. This strategy is particularly relevant for optimizing performance in systems with limited RAM, as it balances the degree of multiprogramming (running multiple processes) and CPU utilization. In [22] proposed page replacement policy monitors the current working-set size and controls the deferring level of dirty pages, preventing excessive preservation that could lead to increased page faults, thus optimizing performance while minimizing write traffic to PCM. In [23] authors modified the ballooning mechanism to enable memory allocation at huge page granularity. Next, they developed and implemented a huge page working set estimation mechanism capable of precisely assessing a virtual machine's memory requirements in huge page-based environments. Integrating these two mechanisms, they employed a dynamic programming algorithm to attain dynamic memory balancing. Also, in [24–26] discussed working set size (WSS) estimation to predict memory demand in virtual machines, which helps optimize memory management. By accurately estimating WSS, the strategy minimizes page faults by ensuring sufficient memory allocation to meet actual usage needs. The working set strategy solves the problem of page faults by preventing actively used pages from being freed, even if the code is suboptimal. However, poor locality increases the size of the working set and makes it too large to fit in RAM, which can negate the benefits of the strategy. We propose an approach to optimize the working set size by using combinatorial space, in the form of Hasse diagram. This problem is classified as *NP*-hard, meaning that finding the optimal solution is computationally hard for large instances, as it would require checking an exponentially large number of possibilities. Thus, the research has also a fundamental aspect [27–31] that has encouraged us in our research efforts.

In this paper, we focus on the problem of page faults minimization for virtual memory systems. Motivated by the need to achieve either an optimal or near-optimal solution, our goal is to construct an approach based on identifing functional and corresponding constraints using the working set swapping strategy to minimize page faults. We use the geometric interpretation of the computational process because it offers a visual and analytical tool for solving a problem that is typically approached through algorithmic or heuristic methods. This approach could potentially reveal patterns or properties not evident in purely computational models.

2. Methods

In computing systems with page-based organization of the virtual memory, programs generate a sequence of references (accesses) to their pages during execution, which we will call as "control state". At any moment of program execution, the physical memory (RAM) does not contain all pages of the program, but only a part of them (the resident set). Figure 1 shows an example of virtual and physical memory. Virtual memory contains blocks of different sizes divided into pages. The physical memory contains copies of the virtual memory pages and here the blocks are restructured. The size of the physical memory at any moment of the computational process is much smaller than the size of the virtual memory. Let the program code with poor locality that requires segmentation consist of n blocks with numbers $i_1, i_2, \dots, i_n$ which are singled out in advance and scattered over p pages $S_1, S_2, \dots, S_p$ of a virtual memory.

 doi:10.20944/preprints202507.0494.v1

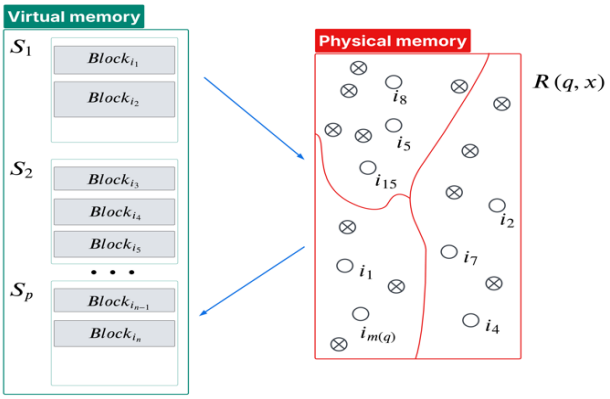


Figure 1. Swapping process.

The code execution causes a problem because of generating a redundant number of page faults, which can be greatly affected by the reason of poorly structured program code and that reduces performances both of the program code and a system itself. Let v_r, be a length of r-th page, $r = 1,2,\dots,p$ and l_i be a length of block i, $i = 1,2,\dots,n$. Thus the system supports multidimensional size of pages [32]. As blocks we mean a part of the code such as subroutines, linear segments of a code, separate interacting programs, data blocks and etc. Distribution of blocks $i_1, i_2, \dots, i_n$ over pages $S_1, S_2, \dots, S_p$ is assigned, for example via Boolean matrix $x = (x_{ri})_{p\times n}$, where an element $x_{ri} = 1$, if the block with number i belongs to the page with number r and $x_{ri} = 0$, otherwise. All of such kind matrices we denote via X.

In our case, working set $R(q, x)$ is generated by control state q and matrix x. As corresponding denotation for control state, we will use q. The control state q_t of the program at moment t, it is a sequence of program references to their pages for the last k moments before moment t. Figure 1 indicated that control state q is $q = (i_1, i_2, \dots, i_{m(q)})$ where i_j is block number $(j = 1,2,\dots,m(q))$, which belongs to q and any of them marked as ○. Another one symbol ⊗ in Figure 1 means blocks (or its numbers) which does not belong to q but belong to corresponding page of working set $R(q, x)$ and present in the physical memory. In other words, all elements ○ are blocks that are often referenced and they form the working set, other elements ⊗ are also blocks that do not form the working set, but they can be present in physical memory at any moment of the computational process. For the matrix $x \in X$, there are constraints (a)-(c) [33], which are described below:

Functional: As a functional of the main problem, we will take a mathematical expectation of number of page faults for one run of the program code. As a functional of the auxiliary problem, we will take a mean value of page faults for $h \geq 1$ runs of the program code.

Constraint (a): Total length of the blocks belong to any page does not exceed the length of this page.

Constraint (b): Any block of the program code belongs only one page of the program code.

Constraint (c): Total length of any working set generated while execution of the program code does not exceed some system constant that known in advance.

Constraints (a)-(c) have to be assigned by a matrix $x = (x_{ri})_{p\times n}$, which defines distribution of the blocks $i_1, i_2, \dots, i_n$ over pages $S_1, S_2, \dots, S_p$. An important role for our consideration plays a Boolean matrix $x = (x_{ri})_{p\times n}$ which determines the structure of a program, i.e., distribution blocks $b_1, b_2, \dots, b_n$ of a program over pages $S_1, S_2, \dots, S_p$. For x it has to hold constraints (a)-(c), and all of such kind of matrices form the set X. Next, we present an example of the structure of matrix $x = (x_{ri})_{p\times n}$ with control state $q = (i_1, i_2, \dots, i_{m(q_t)})$ singled out among columns of matrix $x = (x_{ri})_{p\times n}$. The matrix $x = (x_{ri})_{p\times n}$ helps to calculate the function $\delta_{qi}(x)$:

 doi:10.20944/preprints202507.0494.v1

$x=(x_{ri})_{p\times n}=$

	1	i_1	i_2	...	i ..	$i_{m(q)}$	n
1	0	0	1	...	0..	..	0
2	0	0	0	...	0..	..	0
⋮	0	1	0	...	1..	..	1
	1	0	0	...	0..	..	0
	0	0	0	...	0..	..	0
⋮	0	0	0	...	0..	..	0
p	0	0	0	...	0..	..	0

2.1. Geometric Interpretation of the Computational Process

We consider the geometric interpretation of the computational process using the Hasse diagram, a graphical representation of a partially ordered set [34]. Each element of the set is represented by a node. A line (edge) exists between each pair of nodes b and c such that $b \leq c$ and there is no d such that $b \leq d \leq c$, i.e., we say that c covers b [35]. In combinatorial space the control state q_0 may happen at latter moments when our program unexpectedly offloads from the physical memory and after a while the program activates as if it runs from the start (cold start). Another way to start is a warm start when the system is trying to continue the computing process from level 1 or 2. Further, we propose that any such event should restore as a warm start (restart) and treat it as one additional page fault, which we will take into account in additional expressions (1), (2) for functionals of main and auxiliary problems.

Let the set of control state q be denoted as Q. When the set Q is formed we have to find the subset of the Q, which we denote as $\hat{Q}$, and which will be useful for us under the constraint (c). Any element $\hat{q} \in \hat{Q}$ has the property, namely, in the Q there is no element q such as $\hat{q} \subsetneq q$. Conceptually looking at the Hasse diagram, as shown in Figure 2, the element $\hat{q}$ is a node which is a peak-node under any random walk path over nodes of the Hasse diagram. Following our approach for any sequence of control state already from Q along the axis t with fixed $\theta \in D$ is a corresponding random walk path over nodes of the combinatorial space. Thus, the computational process is a random walk path through the nodes of the combinatorial space, for which we use the Hasse diagram.

A Hasse diagram is a two poles combinatorial space with a number of blocks $n = 6$, $n + 1$ and several levels. The down pole, located at the zero level (0) corresponds to the empty set $q_0 = \emptyset$ for starting any process. The upper pole corresponds to a number of blocks of the program code, i.e., $n = 6$. Elements of the set Q correspond to appropriate nodes of the combinatorial space. Any control state $q = (i_1, i_2, \dots, i_m) \in Q$ that corresponds to the intermediate node $(i_1, i_2, \dots, i_m)$. For example, at level $l(1 \leq l < n)$, is an ordered record, such that $i_1 < i_2 < \cdots < i_m$ and connects with m nodes of the level $l - 1$ and with $n - m$ nodes of the level $l + 1$. And, node $(2,3)$ of the level 2, connects with two nodes at level 1, namely, (2) and (3) and connects also with four nodes at level 3, namely, (1,2,3), (2,3,4), (2,3,5) (2,3,6). Among nodes of the singled out path, black nodes are: (2,3), (1,2.4), (1,3,4,5), (3,6) and are needed for us to optimize nodes $\bar{q}$, which are down nodes of the edges $(\bar{q}, \hat{q})$, namely: (2,4), (1,4,5) (3).

 doi:10.20944/preprints202507.0494.v1

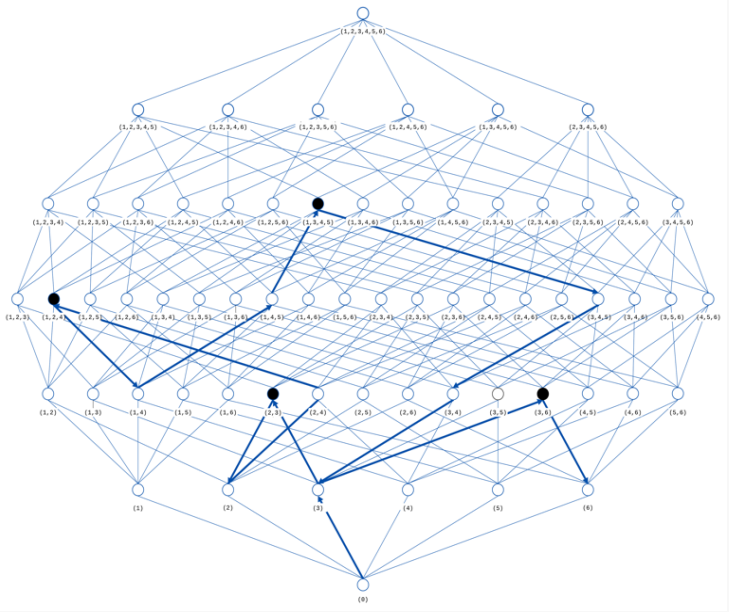


Figure 2. Hasse diagram.

Figure 3 indicates a dedicated random walk path along nodes under $k = 4$, that correspond to a working set and singled out path with $\hat{q}$ and $\bar{q}$ nodes on it. Eventually, we have a random walk path over nodes with $\hat{q}$ and $\bar{q}$. Under multiple runs nodes $\hat{q}$ and $\bar{q}$ can be changed but at all times they are existing in the computional process, including the final situation, when the set Q is determined. The only point to note for description of an algorithm to determine $\hat{Q}$ is very simple and consists of sequentially sorting out elements of the Q and comparison to a current q. First, it takes removal of the current element q from Q. If yes, i.e., then q has to be crossed out of consideration as the candidate for $\hat{q}$. If no then we have to continue the check of an inclusion into the next q from Q. If we cannot find such q from Q and the set Q already is exhausted then q becomes $\hat{q}$ and we add $\hat{q}$ into $\hat{Q}$. We repeat the process with the next elements of Q as q until the set Q is exhausted and we form the set $\hat{Q}$. It has to be noted that any run of the program takes finite time.

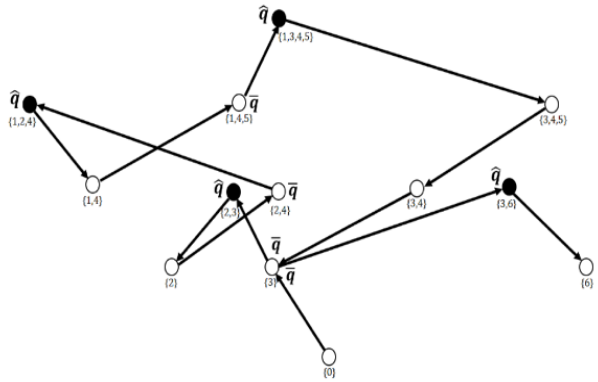


Figure 3. A dedicated random walk path along nodes.

2.2. Functionals $F^0(x)$ and $F^{(h)}(x)$. Constraints for $x \in X$.

In this subsection we describe functionals and constrains of the main and auxiliary problems. We will start by finding expressions of the functional of both the main and auxiliary problem and expressions for corresponding constrains for matrix $x = (x_{ri})_{p\times n}$. As well, it is useful to determine the connection between the calculation that will be done and its geometric interpretation. Using information given above, we introduce a random variable ξ_{qi} which is a number of references to

 doi:10.20944/preprints202507.0494.v1

block i under the execution of control state q for one run of the program. Let random variable ξ_{qi}^{j} be the same as ξ_{qi} but in j-th run of the program $j = 1,2,\dots,h$. Let expected value of ξ_{qi} and ξ_{qi}^{j} be:

$$E(\xi_{qi}) = E(\xi_{qi}^{j}) = E_{qi},\ j = 1,2,\dots,h$$

and a mean value:

$$E_{qi}^{(h)} = (1/h)\sum_{j=1}^{h}\xi_{qi}^{(j)},\ \text{for any}\ q \in Q, i = 1,2,\dots,n.$$

Calculation of the $\delta_{qi}(x)$ can be done in the following way:

$$\delta_{qi}(x) = \begin{cases} 0, & \text{if block } i \in S \in R(q,x) \\ 1, & \text{otherwise} \end{cases}$$

In other words, the value $\delta_{qi}(x) = 0$ is corresponding to the absence of the page fault under event $q \to i$, the value $\delta_{qi}(x) = 0$, if block i belongs to some page S from $R(q,x)$. Otherwise the value $\delta_{qi}(x) = 1$ is corresponding to the page fault. If block $i \in q$ then it has to be $\delta_{qi}(x) \equiv 0$ for any $x \in X$. Next, we can remove the control state q_0 from Q, then a total number of page faults for one run of the program will be:

$$\xi = \sum_{q\in Q.}\sum_{i=1}^{n}\xi_{qi}\cdot\delta_{qi}(x) + \sum_{i=1}^{n}\xi_{q_0 i}$$

and for the functional of the main problem which has to be minimized we have:

$$F^{0}(x) = \sum_{q\in Q.}\sum_{i=1}^{n}E_{qi}\cdot\delta_{qi}(x) + \sum_{i=1}^{n}E_{q_0 i} \longrightarrow \min_{x\in X} \tag{1}$$

It is worth noting that in the expression for ξ the any value ξ_{qi} does not depend on matrix $x \in X$ and quite the opposite the function $\delta_{qi}(x)$ depends on given $q \in Q$ and i and $x \in X$ and does not depend on random event $q \to i$ and where it happens. For the functional $F^{(h)}(x)$ of the auxiliary problem holds:

$$F^{(h)}(x) = \sum_{q\in Q.}\sum_{i=1}^{n}E_{qi}^{(h)}\,\delta_{qi}(x) + \sum_{i=1}^{n}E_{q_0 i}^{(h)} \longrightarrow \min_{x\in X} \tag{2}$$

It is interesting to note that value $E_{qi}^{(h)}$ from (2) can be assigned to the edge that connects the node q and the node $q \cup i$ in the Hasse diagram, where the function $\delta_{qi}(x) = 1$ and otherwise. This edge has to be weighted as zero if the function $\delta_{qi}(x) = 0$. It may help to calculate the value of the functional $F^{(h)}(x)$ for fixed $x \in X$. It will be sufficient to determine whether the weight of any edge in question is 1 or 0, which means whether a page error has occurred or not. The system of constraints (a) - (c) setting the set of X of admissible solutions for both the main problem (1) and for auxiliary problem (2) registers in the form:

$$\sum_{i=1}^{n} l_i \cdot x_{ri} \le v_r, \qquad r = 1,2,\dots,p; \tag{3}$$

$$\sum_{r=1}^{p} x_{ri} = 1, \qquad i = 1,2,\dots,n; \tag{4}$$

 doi:10.20944/preprints202507.0494.v1

$$\sum_{r=1}^{p} v_r \cdot H_{qr}(x) \le N_q, \qquad q \in Q; \tag{5}$$

$$x_{ri} \in \{0,1\}, r = 1,2,\dots,p; i = 1,2,\dots,n \tag{6}$$

where in (5) the value v_r is length of page $r, r = 1,2,\dots,p$. The system (3)-(6) contains $p + n + |Q|$ non-trivial correlations. Note that constraints (3)-(5) correspond to constraints (a)-(c) respectively. The function $H_{qr}(x)$: $H_{qr}(x) = 1$, if page $S_r \in R(q,x)$ and $H_{qr}(x) = 0$, otherwise, i.e., the function $H_{qr}(x)$ is the characteristic function of the $R(q,x)$. Under given q and r it is easy to calculate $H_{qr}(x)$ via elements of the matrix x, namely if $q = (i_1, i_2, \dots, i_{m(q)}) \in Q$ then:

$$H_{qr}(x) = \max_{1 \le j \le m(q)} x_{ri_j}.$$

2.3. Reduction in a Number of Inequalities of the Control State in Working Set $R(q,x)$

Constraint (5) contains $|Q|$ inequalities and probably there are a lot. Here is an opportunity to reduce essentially a number of inequalities in (5). As already mentioned, from a practical point of view, we may propose that there exists a system constant, let it be N, which limits the dimension of any working set $R(q,x)$ and which is known in advance. It is necessary to note the set $\hat{Q}$ and then we can substitute system (5) for:

$$\sum_{r=1}^{p} v_r \cdot H_{\hat{q}r}(x) \le N, \qquad \hat{q} \in \hat{Q}; \tag{7}$$

but first we must put in (5) for all $N_q = N$, $q \in Q$. Let $|R(q,x)|$ be the length of the working set $R(q,x)$ $q \in Q$, and $x \in X$. To give a ground for substitution it is worth paying attention to Figure 2 and Figure 3 with black nodes on them corresponding to control state $\hat{q} \in \hat{Q}$ and nodes $q \in Q$, such as $q \subseteq \hat{q}$. Then the next correlations hold: if $q \subseteq \hat{q}$ then $R(q,x) \subseteq R(\hat{q},x)$ and $|R(q,x)| \le$, where $q \in Q$, $\hat{q} \in \hat{Q}$ and if inequality (7) holds for some $\hat{q} \in \hat{Q}$ then it also holds for any $q \in Q$ which $q \subseteq \hat{q}$. Here it is taken into account that any $q \in Q$ belongs to at least one $\hat{q}$ from $\hat{Q}$ as shown in Figure 3, node (3). Evidently, the set X of admissible solutions is non empty since an initial distribution block $i_1, i_2, \dots, i_n$ over pages $S_{g_1}, S_{g_2}, \dots, S_{g_p}$ satisfies constraints (3)-(6).

3. Results

The nonlinear model of the reorganization of the program code which is constructed above, contains nonlinear functional (1) and/or (2) and both linear inequalities (3), (4) and nonlinear system of constraints in (5). The power of the set $\hat{Q}$ in (7) is not too large in contrast with the power of Q in (5). The constraints (5), (7) show instead of controlling a size of any $R(q,x)$ with totally $|Q|$ inequalities, after substitution (7) instead of (5) we have in (7) only $|\hat{Q}|$ inequalities. As for functional (1) or (2) it can be reduced to a number of addends in (1) or (2) on the basis of the idea that if block $i \in q$ then $\delta_{qi}(x) \equiv 0$ for any $x \in X$ and second sum in (1) or (2) has instead of $i = 1,2,\dots,n$, only the indexes $i \in I(q)$, where the set $I(q)$ does not contain such i belongs to q. Under given $q = (i_1, i_2, \dots, i_{m(q)})$ and i and under the event $q \to i$, if $i_1 \notin (i_2, i_3, \dots, i_{m(q)})$ and $i \in (i_2, i_3, \dots, i_{m(q)})$ then there will be no page fault. If $i_1 \in (i_2, i_3, \dots, i_{m(q)})$ and $i \in (i_2, i_3, \dots, i_{m(q)})$ then there will also be no page fault. It is important to note that some methods of discrete optimization, based on construction of valuation function, in our case, for the problem (2) on the basis of geometric interpretation of the computational process, it is the lower valuation function, which is written on the left side of (8):

$$\sum_{\bar{q} \in \hat{Q}} \sum_{\bar{q} \in Q_{\bar{q}}} E^{(h)}_{\bar{q} i_{\hat{q} \setminus \bar{q}}} \cdot \delta_{\bar{q} i_{\hat{q} \setminus \bar{q}}}(x) \le \sum_{q \in Q.} \sum_{i=1}^{n} E^{(h)}_{qi} \cdot \delta_{qi}(x) \tag{8}$$

 doi:10.20944/preprints202507.0494.v1

From [19,36] it follows that, if it is possible to solve the problem with valuation function, which has been written also in (9)

$$\sum_{\hat{q}\in\hat{Q}}\sum_{\bar{q}\in Q_{\hat{q}}} E^{(h)}_{\bar{q}i_{\hat{q}\setminus\bar{q}}}\cdot\delta_{\bar{q}i_{\hat{q}\setminus\bar{q}}}(x)\longrightarrow\min_{x\in X} \tag{9}$$

then it gives the opportunity, with appropriate complexity, to get an exact (optimal) solution of the problem (2) with functional on the right side of (8). The set $Q_{\hat{q}}$ on the left part of (8) is a subset of Q, which is defined by a separate $\hat{q}$ and consists of a number of $\bar{q}\in Q$. If we look to Figure 3, a node $\bar{q}$ has to be connected with node $\hat{q}$ by the edge, i.e., $(\bar{q},\hat{q})$ which is the oriented edge with nodes $\bar{q}$ and $\hat{q}$. Meanwhile it is not necessary to take into account both on the left side of (8) and (9) the edge $(\hat{q},\bar{q})$, since the weight of the $(\hat{q},\bar{q})$ equals 0. On the left side of (8) for any $\hat{q}\in\hat{Q}$ a node $\bar{q}$ is running for edge $(\bar{q},\hat{q})$ until the set $Q_{\hat{q}}$ is exhausted. We include a node $\bar{q}$ into $Q_{\hat{q}}$ if there is at least one reference, while h runs of the program, from node $\bar{q}$ to the node $\hat{q}$. The number i on the left part of (8) is defined as $\hat{q}\setminus\bar{q}$, i.e., $i=\hat{q}\setminus\bar{q}$, let it be a denotation $i_{\hat{q}\setminus\bar{q}}$. The left part of (8) contains a lesser number of addends than the right side of (8). The same we may say about the functional of the problem (1), i.e., about $F^0(x)$. As for optimal solution of (1) it is interesting to point out conditions for initial data when the optimal solution of problem (2), let it be the matrix x_h^*, will be an ε - optimal solution of the problem (1) in sense:

$$Pr\{|F^0(x^*)-\mathrm{F}^0(x_h^*)|\le\varepsilon\}\ge 1-\eta,\quad \varepsilon>0,\ \eta\in(0,1) \tag{10}$$

where the matrix x^* is an unknown optimal solution of the problem (1). Those conditions first of all imply, to determine common properties of the distribution laws for the variables $\xi_{qi}, i=1,2,\dots,n; q\in Q$ and lower bound for the number h of executions (runs) of the code, under which inequality (10) holds.

Under the known values E_{qi} in (1) i.e., the distribution law of each random variable $\xi_{qi}, i=1,2,\dots,n;\ q\in Q$ is known, the algorithm of the solution, both the initial problem (1), and the problem (2) can be based on valuation function (see (8)) and the property of the function $\delta_{qi}(x)$:

$$\delta_{\bar{q}'i}(x)+\delta_{\bar{q}''i}(x)\le\delta_{\bar{q}'\cap\bar{q}''i}(x)+\delta_{\bar{q}'\cup\bar{q}''i}(x),$$

which takes place for any $\bar{q}',\bar{q}''\in\hat{Q}\subset Q$ and represents a special case of the property of supermodularity. In the case of unknown values E_{qi} (the distribution law of the random variables $\xi_{qi}, i=1,2,\dots,n; q\in Q$ is unknown) the situation for solution of problem (1) becomes more complicated. In this case the problem (2) could be used as an auxiliary problem for (1) and optimal solution of (2), i.e., x_h^* can be taken as the solution of (1) in the sense of the inequality (10).

We used a heatmap technique to visualize page faults and different working set sizes as shown in Figure 4(a), (b) and (c), where are shown the number of page faults with a different working set size and a fixed Physical Memory Size. Figure 4(d) indicates a comparison result of page faults simulation in different algorithms like Working Set, LRU (Least Recently Used) and FIFO (First In, First Out). Experiments were performed with different parameters: Virtual Memory Size, Physical Memory Size, Working Set Size, Access Sequence Length, Locality Factor. The Working Set algorithm tries to keep only actively used pages in memory. If the working set size is smaller than the physical memory size, the number of page faults will be low, if working set size increases, then page faults also increase accordingly. The LRU algorithm unloads the page that has not been used for the longest time. It performs well, especially when there is locality in the order of accesses.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 July 2025 doi:10.20944/preprints202507.0494.v1

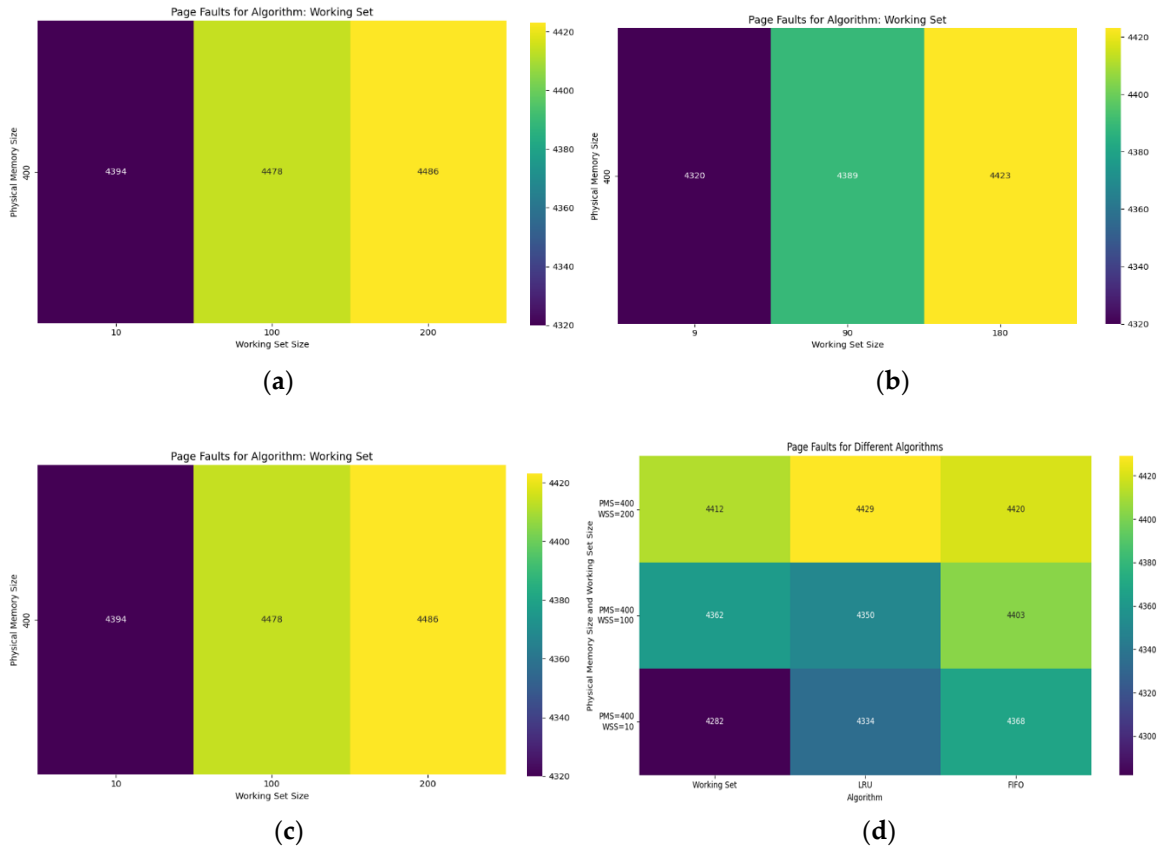


Figure 4. Comparing simulation results of page faults with different working set size in (**a**), (**b**), (**c**) and algorithms in (**d**).

The FIFO algorithm unloads the page that was loaded first. It shows worse results compared to LRU and working set, especially in the presence of cyclic access patterns. The simulation results on page fault optimization show that in the fixed size of physical memory, there are fewer page faults for all algorithms. This is because more pages can be loaded simultaneously, reducing the need for paging. When the physical memory size is small, the number of faults increases because the algorithms are forced to offload pages more frequently. The heatmap shows that for a given algorithm and physical memory size, if the cell color is light, it indicates a large number of page faults. For example, a FIFO with a small physical memory size may have a high number of page faults. If the cell color is dark, it indicates a low number of page faults. For example, an LRU with a large physical memory size may have low page fault values.

Table 1 presents that Working Set is the most effective when there is locality in the order of accesses, especially when the physical memory size is large. LRU performs consistently well and is a compromise between implementation complexity and efficiency. This algorithm effectively minimizes the number of page faults. FIFO is the least efficient, especially when the physical memory size is small, because it does not consider locality and may discard pages that will be used soon.

Table 1. Average page faults for different algorithms.

Algorithm	Average Page Faults	Average Locality Factor
Working Set	4352	0.72
LRU	4371	0.72
FIFO	4397	0.72

We conducted two more experimental studies using a 10-node FPGA-based system arranged in a ring topology. These experiments extend the original analysis by comparing caching algorithms

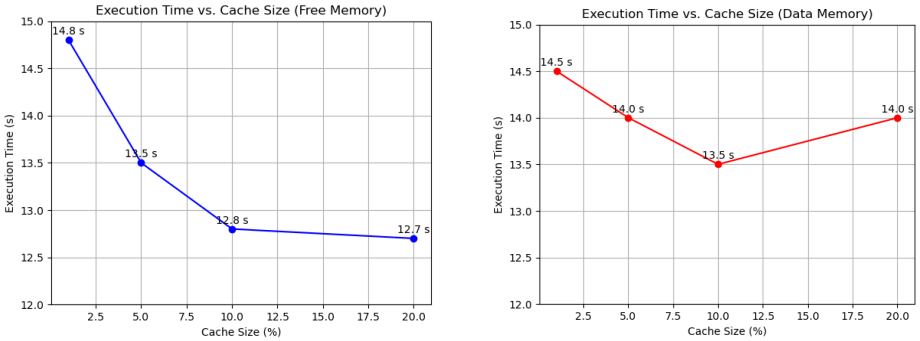
 doi:10.20944/preprints202507.0494.v1

and exploring the impact of varying cache sizes, using a random memory access workload to simulate a worst-case scenario. We evaluated WS, LRU and FIFO caching algorithms with a cache size of 5% allocated from free memory. Each of the 10 nodes performed 100,000 operations, with 80% accessing local memory and 20% requesting remote data. Performance was assessed through average page fault time, page fault frequency, and total execution time. Table 2 shows the results indicate that the WS algorithm consistently outperformed the alternatives. WS reduced the average page fault time to 6 µs and the fault frequency to 35%, yielding a total execution time of 13 seconds. LRU and FIFO showed similar performance, with average fault times of 7 µs and frequencies around 38%, resulting in execution times of 14.5 seconds. Random Replacement performed the worst, with a fault time of 8 µs, a frequency of 40%, and an execution time of 15 seconds.

Table 2. Performance Metrics for Caching Algorithms.

Algorithm	Av. Page Fault Time (µs)	Page Fault Frequency (%)	Total Execution Time (s)
WS	6	35	13
LRU	7	38	14.5
FIFO	7	38	14.5

The superior performance of WS likely stems from its ability to maintain a set of actively used pages, adapting even to random access patterns. LRU and FIFO, while effective in workloads with locality, offer little advantage here, and RR's lack of pattern consideration makes it the least efficient. We varied the cache size—1%, 5%, 10%, and 20% of total memory—using the Working Set algorithm, sourcing memory from both free reserves and data memory (overflow). The same workload and system configuration were applied. As cache size increased from 1% to 10%, page fault frequency dropped from 39% to 33%, and total execution time decreased from 14.8 seconds to 12.8 seconds as shown in Figure 5a. Beyond 10%, at 20%, the frequency stabilized at 32%, and execution time plateaued at 12.7 seconds, indicating diminishing returns. Average page fault time remained steady at around 6 µs across all sizes. With data memory caching, fault frequency decreased from 38% at 1% to 30% at 10%, with execution time improving from 14.5 seconds to 13.5 seconds. However, at 20%, execution time rose to 14 seconds despite a frequency of 29%, as reduced data memory led to more frequent evictions and reloads as shown Figure 5b.



(**a**) (**b**)

Figure 5. (**a**) Execution Time vs. Cache Size (Free Memory); (**b**) Execution Time vs. Cache Size (Data Memory).

These experiments reaffirm the efficacy of RAM page caching, with the Working Set algorithm proving optimal for random workloads. The choice of algorithm significantly impacts performance, with WS reducing execution time by up to 14% compared to no caching (15 seconds).

4. Discussion

The results of the simulation experiments show that the locality of accesses to memory pages is a key factor for increasing the efficiency of paging algorithms. We see that Working Set and LRU are

 doi:10.20944/preprints202507.0494.v1

the most efficient paging algorithms, while FIFO is inferior to them in all parameters. Cache size optimization reveals a sweet spot around 10% for both free and data memory caching. Beyond this, free memory caching yields minimal gains, while overflow caching introduces trade-offs due to limited data space. This suggests a need for careful memory allocation to balance caching benefits and data availability. Future research could investigate dynamic cache sizing algorithms that adapt to workload changes or explore realistic application-specific workloads to broaden the applicability of these findings.

5. Conclusions

Poor code locality encourages the study of program restructuring and optimization strategies, such as the working set strategy, to minimize page faults in[virtual memory systems. This paper proposes a combinatorial approach to optimize working set size, and aims at obtaining near optimal solutions using geometric interpretation and functional constraints. In order to optimize the calculations, a valuation function was found which includes the empirical average of the experiments and a system of constraints. Our proposed geometric interpretation of the computational process in the form of a Hasse diagram helps to reduce the dimensionality of the page faults minimization problem. The approach outlined in the paper provides a basis for finding the optimal solution to the main problem, i.e., to page faults minimization, if the minimum distribution of random variables is known in advance. Otherwise, with an unknown minimum of the random variable distribution, the constructed approach provides the basis for finding an accurate solution to the auxiliary problem, i.e., the experimental average of page faults.

Currently, the working set strategy is usually used as a theoretical research base, either for comparison or for auxiliary purposes, as it is considered expensive to implement. However, in our case, if the program code has a block structure, then the results obtained can be used to build a fast, accessible and affordable swap algorithm.

Author Contributions: Conceptualization, A.M., G.B. and Z.S.; methodology, A.M.; software, A.M. and A.B; validation, Z.S., A.B. and G.A.; formal analysis, G.B., G.A.; investigation, G.A.; resources, A.M.; data curation, A.B. and Z.S.; writing—original draft preparation, A.M., G.A. and G.B.; writing—review and editing, A.M. and Z.S.; visualization, A.B. and A.M.; supervision, A.M.; project administration, G.B.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by a grant from the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan, grant number “AP19174930”.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data can certainly be provided upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Allen, T.; Cooper, B.; Ge, R. Fine-Grain Quantitative Analysis of Demand Paging in Unified Virtual Memory. *ACM Trans. Archit. Code Optim.* **2024**, *21*, 1–24, doi:10.1145/3632953.
2. Nestor, J.; Yin, Z. Work in Progress: A Visualization Aid for Learning Virtual Memory Concepts, in Proceedings of ASEE Annual Conference and Exposition: Excellence Through Diversity (ASEE), Minneapolis, 2022.
3. Lian, Z.; Li, Y.; Chen, Z.; Shan, S.; Han, B.; Su, Y. EBPF-Based Working Set Size Estimation in Memory Management. In Proceedings of the 2022 International Conference on Service Science (ICSS); IEEE: Zhuhai, China, 2022; pp. 188–195.
4. Chen, Y.-C.; Wu, C.-F.; Chang, Y.-H.; Kuo, T.-W. Exploring Synchronous Page Fault Handling. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *41*, 3791–3802, doi:10.1109/TCAD.2022.3197517.

 doi:10.20944/preprints202507.0494.v1

5. Doron-Arad, I.; Naor, J. (Seffi) Non-Linear Paging. *LIPIcs, Volume 297, ICALP 2024* **2024**, *297*, 57:1-57:19, doi:10.4230/LIPICS.ICALP.2024.57.
6. Wood, C.; Fernandez, E.B.; Lang, T. Minimization of Demand Paging for the LRU Stack Model of Program Behavior. *Information Processing Letters* **1983**, *16*, 99–104, doi:10.1016/0020-0190(83)90034-0.
7. Dyusembaev, A. E. On one approach to the problem of segmenting programs, Doklady Akademii Nauk, vol. 329, no. 6, pp. 712-723, 1993.
8. Teabe, B.; Yuhala, P.; Tchana, A.; Hermenier, F.; Hagimont, D.; Muller, G. Memory Virtualization in Virtualized Systems: Segmentation Is Better than Paging 2020.
9. Ngetich, M.K.Y.; Otieno, C.; Kimwele, M.; Gitahi, S. Advancements in Code Restructuring: Enhancing System Quality through Object-Oriented Coding Practices. In Proceedings of the 2023 IEEE 27th International Conference on Intelligent Engineering Systems (INES); IEEE: Nairobi, Kenya, July 26 2023; pp. 000125–000130.
10. Yegon Ngetich, M.K.; Otieno, D.C.; Kimwele, D.M. A Model for Code Restructuring, A Tool for Improving Systems Quality In Compliance With Object Oriented Coding Practice. *IJCATR* **2019**, *8*, 196–200, doi:10.7753/IJCATR0805.1010.
11. Peachey, J.B.; Bunt, R.B.; Colbourn, C.J. Some Empirical Observations on Program Behavior with Applications to Program Restructuring. *IIEEE Trans. Software Eng.* **1985**, *SE-11*, 188–193, doi:10.1109/TSE.1985.232193.
12. D.B. Roberts, Practical Analysis for Refactoring (1999). (Ph.D. thesis) University of Illinois
13. Cedrim, D.; Garcia, A.; Mongiovi, M.; Gheyi, R.; Sousa, L.; De Mello, R.; Fonseca, B.; Ribeiro, M.; Chávez, A. Understanding the Impact of Refactoring on Smells: A Longitudinal Study of 23 Software Projects. In Proceedings of the Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering; ACM: Paderborn Germany, August 21 2017; pp. 465–475.
14. Agnihotri, M.; Chug, A. Severity Factor (*SF*): An Aid to Developers for Application of Refactoring Operations to Improve Software Quality. *J Software Evolu Process* **2024**, *36*, e2590, doi:10.1002/smr.2590.
15. Agnihotri, M.; Chug, A. Understanding the Effect of Batch Refactoring on Software Quality. *Int J Syst Assur Eng Manag* **2024**, *15*, 2328–2336, doi:10.1007/s13198-023-02247-x.
16. Coelho, F.; Massoni, T.; L.G. Alves, E. Refactoring-Aware Code Review: A Systematic Mapping Study. In Proceedings of the 2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR); May 2019; pp. 63–66.
17. Arasteh, B.; Ghanbarzadeh, R.; Gharehchopogh, F.S.; Hosseinalipour, A. Generating the Structural Graph-based Model from a Program Source-code Using Chaotic Forrest Optimization Algorithm. *Expert Systems* **2023**, *40*, e13228, doi:10.1111/exsy.13228.
18. Arasteh, B.; Abdi, M.; Bouyer, A. Program Source Code Comprehension by Module Clustering Using Combination of Discretized Gray Wolf and Genetic Algorithms. *Advances in Engineering Software* **2022**, *173*, 103252, doi:10.1016/j.advengsoft.2022.103252.
19. A. E. Dyusembaev, "Mathematical models of program segmentation," M: Fizmatlit (Nauka, MAIK), 2001.
20. Cheng, W.; Wu, C.-F.; Chang, Y.-H.; Lin, I.-C. GraphRC: Accelerating Graph Processing on Dual-Addressing Memory with Vertex Merging. In Proceedings of the Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design; ACM: San Diego California, October 30 2022; pp. 1–9.
21. Denning, P.J. The Working Set Model for Program Behavior. In Proceedings of the Proceedings of the ACM symposium on Operating System Principles - SOSP '67; ACM Press: Not Known, 1967; p. 15.1-15.12.
22. Park, Y.; Bahn, H. A Working-Set Sensitive Page Replacement Policy for PCM-Based Swap Systems. *JSTS:Journal of Semiconductor Technology and Science* **2017**, *17*, 7–14, doi:10.5573/JSTS.2017.17.1.007.
23. Sha, S.; Hu, J.-Y.; Luo, Y.-W.; Wang, X.-L.; Wang, Z. Huge Page Friendly Virtualized Memory Management. *J. Comput. Sci. Technol.* **2020**, *35*, 433–452, doi:10.1007/s11390-020-9693-0.
24. Hu, J.; Bai, X.; Sha, S.; Luo, Y.; Wang, X.; Wang, Z. Working Set Size Estimation with Hugepages in Virtualization. In Proceedings of the 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social

 doi:10.20944/preprints202507.0494.v1

Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom); IEEE: Melbourne, Australia, 2018; pp. 501–508.

25. Nitu, V.; Kocharyan, A.; Yaya, H.; Tchana, A.; Hagimont, D.; Astsatryan, H. Working Set Size Estimation Techniques in Virtualized Environments: One Size Does Not Fit All. In Proceedings of the Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems; ACM: Irvine CA USA, June 12 2018; pp. 62–63.
26. Verbart, A.; Stolpe, M. A Working-Set Approach for Sizing Optimization of Frame-Structures Subjected to Time-Dependent Constraints. *Struct Multidisc Optim* **2018**, *58*, 1367–1382, doi:10.1007/s00158-018-2063-7.
27. Dyusembaev, A.E. On the Correctness of Algebraic Closures of Recognition Algorithms of the "Tests" Type. *USSR Computational Mathematics and Mathematical Physics* **1982**, *22*, 217–226, doi:10.1016/0041-5553(82)90111-2.
28. Alanko, T.O.; Haikala, I.J.; Kutvonen, P.H. Program Restructing in Segmented Virtual Memory. *Performance Evaluation* **1981**, *1*, 153–169, doi:10.1016/0166-5316(81)90017-1.
29. Pâris, J.-F.; Ferrari, D. An Analytical Study of Strategy-Oriented Restructuring Algorithms. *Performance Evaluation* **1984**, *4*, 117–132, doi:10.1016/0166-5316(84)90006-3.
30. Ghosal, D.; Serazzi, G.; Tripathi, S.K. The Processor Working Set and Its Use in Scheduling Multiprocessor Systems. *IIEEE Trans. Software Eng.* **1991**, *17*, 443–453, doi:10.1109/32.90447.
31. Marshall, W.T.; Nute, C.T. Analytic Modelling of "Working Set like" Replacement Algorithms. In Proceedings of the Proceedings of the 1979 ACM SIGMETRICS conference on Simulation, measurement and modeling of computer systems - SIGMETRICS '79; ACM Press: Boulder, Colorado, United States, 1979; pp. 65–72.
32. Denning, P.J. Working Set Analytics. *ACM Comput. Surv.* **2021**, *53*, 1–36, doi:10.1145/3399709.
33. Dyusembaev, A. E. Correct models of program segmenting. Journal of pattern recognition and image, Analises USA, vol. 3, no. 6, 1993, pp.187-204.
34. Church, A. Garrett Birkhoff. Lattice Theory. Revised Edition. American Mathematical Society Colloquium Publications, Vol. 25. American Mathematical Society, New York1948, Xiii + 283 Pp. *J. symb. log.* **1950**, *15*, 59–60, doi:10.2307/2268441.
35. Mrena, M.; Kvassay, M. Generating Monotone Boolean Functions Using Hasse Diagram. In Proceedings of the 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS); IEEE: Dortmund, Germany, September 7 2023; pp. 793–797.
36. Kochetov, Yu.; Pljasunov, A.V. Genetic local search for the graph partitioning problem under cardinality constraints, Zh. Vychisl. Mat. Mat. Fiz., vol. 52, no. 1, pp. 164-176, 2012.