АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ
ALGORITHMIC AND PROGRAMMING

УДК 004.4-004.9

A. A. Prihozhy
Belarusian National Technical University

OPTIMIZATION OF PROGRAMMING TEAMS ON COMPATIBILITY OF PROGRAMMERS

The programming team formation problem has been solved using different optimization criteria: programmer and programming team competences; required set of skills, productivity of teams, etc. This paper formulates the problem of optimizing programming teams accounting for pairwise compatibility of programmers described by a matrix whose elements are changes of the programmer and team runtimes when two programmers are included in the same team. When the matrix element is positive the runtime increases, when it is negative the runtime decreases. The problem is formulated as to partition a set of programmers into a set of teams in such a way that the overall teams' runtime is minimal. The graph clique partitioning problem is related to the team formation problem. It maximizes the overall sum of constant weights of edges located within the cliques. The team formation problem differs because it searches for a solution by changing the graph edge weights. Both problems are NP-hard. The paper proposes a greedy algorithm of stepwise pairwise merge of programming teams and provides a software for team optimization. Experimental results show that the algorithm finds partitions of large sets of programmers and generates teams which reduce the runtime by up to 36 % compared to the one-programmer teams and the single team.

Keywords: programmer, compatibility of programmers, team formation problem, project, runtime, optimization.

For citation: Prihozhy A. A. Optimization of programming teams on compatibility of programmers. *Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics*, 2023, no 2 (272), pp. 104–110. DOI: 10.52065/2520-6141-2023-272-2-15.

А. А. Прихожий
Белорусский национальный технический университет

ОПТИМИЗАЦИЯ ПРОГРАММИСТСКИХ КОМАНД ПО СОВМЕСТИМОСТИ ПРОГРАММИСТОВ

Задача формирования команд программистов решалась с использованием различных критериев оптимизации: компетенций программиста и команды программистов; требуемого набора навыков, производительности команд и др. В статье формулируется задача оптимизации команд с учетом попарной совместимости программистов, описываемых матрицей, элементами которой являются изменения времен работы программистов и команд при попарном включении программистов в одну команду. Если элемент матрицы положительный, время работы увеличивается, если отрицательный, время работы уменьшается. Задача формулируется так, чтобы разделить множество программистов на множество команд таким образом, чтобы общее время работы команд было минимальным. Задача кликового разбиения графа связана с задачей формирования команд. Она максимизирует общую сумму постоянных весов ребер в кликах графа. Задача формирования команд отличается тем, что она ищет решение, изменяя веса ребер графа. Обе задачи являются NP-трудными. В статье предлагается жадный алгоритм пошагового слияния команд и разрабатывается программное обеспечение для оптимизации команд. Экспериментальные результаты показали, что алгоритм находит разбиение больших множеств программистов и генерирует команды, сокращающие время работы до 36% по сравнению с командами из одного программиста и единой командой.

Ключевые слова: программист, совместимость программистов, задача формирования команд, проект, время выполнения, оптимизация.

Для цитирования: Прихожий А. А. Оптимизация программистских команд по совместимости программистов // Труды БГТУ. Сер. 3. Физико-математические науки и информатика. 2023. № 2 (272). С. 104–110. DOI: 10.52065/2520-6141-2023-272-2-15 (In English).

Introduction. The problem of allocating programmers to programming teams has not received too much attention in the scientific literature. Agile [1] is a set of values and principles of developing software and finding solutions over joint efforts of development teams and customers. Work [1] describes the process of task allocation as including three mechanisms of workflow across teams and five types of task allocation strategies.

Work [2] emphasizes that a successful software development team must be made up of competent developers. Competency being the ability of developers to carry out a job properly is considered as a combination of knowledge, skills and attitudes used to improve performance. The agile team allocation is a NP-hard problem since it comprises the allocation of self-organizing and cross-functional teams. Work [2] presents a hybrid approach based on *NSGA*-II multiobjective metaheuristic and Mamdani Fuzzy Inference Systems to solve the agile team allocation problem.

Agent-based evolutionary methods of optimization [3] aim at performing the management of teams. Papers [4–6] propose tools that increase the productivity and efficiency of teams working on various projects. *Wrike* [4] is a platform that manages projects, organizes work, enhances collaboration and accelerates execution across all departments. *Flow* [5] is a modern software for teams, which brings together tasks, projects, timelines, and conversations.

Work [7] proposes a method of formalization and evaluation of the programmers' and programming teams' competency. Work [8] solves the problem of allocating experts to maximum set of programming teams. Since the problem of distributing programmers on a set of teams is combinatorial, works [9–11] developed a genetic-algorithm-based approach to finding problem solutions at different requirements to programmers' competences.

This paper considers how the compatibility of programmers influence the optimization of teams' runtime. Its contribution is as follows:

1. A matrix of compatibility of programmers is proposed which allows the evaluation of changes in the teams' overall runtime.

2. It is shown how the changes in the team counts, sizes and staff influence the teams' runtime.

3. A greedy algorithm of stepwise pairwise merge of teams is developed which exploits programmers' compatibility and minimizes the teams' runtime.

4. The experimental results obtained show that the minimum value of teams' runtime depends on the features of compatibility matrix and the number of programmers.

Main part. *Modelling compatibility of programmers in teams*. Let $P = \{p_0, \ldots, p_{n-1}\}$ be a set of n programmers participating in an IT project. Vector $t = (t_0, \ldots t_i, \ldots t_{n-1})$ describes the basic runtimes in days to be spent by the programmers while working on the project. It does not account the interaction of programmers within a team.

Let $G = \{g_1 \ldots g_k\}$ be a set of teams the programmers are allocated to. If the programmers are included and work in the same team, their runtimes are to be corrected. The correction depends on the compatibility of programmers. Matrix dP represents in percent changes in the programmers' runtimes. Its diagonal values are $dP_{i,i} = t_i$. Its nondiagonal values $dP_{i,j}$ describe changes in the runtime of programmer j caused by programmer i. If $dP_{i,j}$ is negative, the runtime spent by programmer j decreases, if the value is positive, the runtime increases. The dP matrix describes both values $dP_{i,j}$ and $dP_{j,i}$, which are different in general case. Regarding combinations of their signs, we consider four situations:

1. Both $dP_{i,j}$ and $dP_{j,I}$ are negative. It means that t_i and t_j are reduced due to influence of the p_i and p_j programmers each other when, for example, they exchange knowledge and experience on technologies needed for the project.

2. Value of $dP_{i,j}$ is negative and value of $dP_{j,i}$ is positive. It means that t_i is increased and t_j is decreased, which can happen when, for example, programmer p_i transfers knowledge and experience to programmer p_j and spend some time to do it without getting knowledge and experience in opposite direction.

3. Value of $dP_{i,j}$ is positive and value of $dP_{j,i}$ is negative. It means that t_j is increased and t_i is decreased when programmer p_j transfers knowledge and experience to programmer p_i without getting help in opposite direction.

4. Both $dP_{i,j}$ and $dP_{j,i}$ are positive. It means that t_i and t_j are both increased since the p_i and p_j programmers are incompatible within one team.

Matrix dT represents programmer runtime changes in days (may be in months) and is calculated over matrix dP. Its nondiagonal element dT_{ij} is

$$dT_{i,j} = t_i \cdot dP_{i,j} / 100. \quad (1)$$

If programmer j is included in team g, its runtime t_j is changed to $t_j(g)$ that is evaluated as

$$t_j(g) = t_j + \sum_{i \in g, i \neq j} dT_{i,j} = t_j \left(1 + dT_j(g)\right), \quad (2)$$

where

$$dT_j(g) = \sum_{i \in g, i \neq j} \left(dP_{i,j} / 100\right). \quad (3)$$

The overall runtime $T(g)$ of the programmers in team g is

$$T(g) = \sum_{i \in g} t_j(g) \quad (4)$$

and the overall runtime of set G of teams is

$$T^G = \sum_{g \in G} T(g). \quad (5)$$

The compatibility of each programmer with all other programmers can be evaluated when all the programmers are included in team *single* = *P*. In the team, the changed runtime $t_j(single)$ of programmer j is calculated with (2) and is assigned to the matrix element dT_{jj}. In the paper, we calculate the dT matrix and carry out the optimization of each team's staff, if $dT_j(g)$ calculated with (3) is not less than –1. Since both elements $dT_{i,j}$ and $dT_{j,i}$ are used simultaneously when i and j are included in the same team, we create a matrix dS where $dS_{i,j} = dT_{i,j} + dT_{j,i}$, $i < j$, and $dS_{i,j} = 0$ otherwise.

To illustrate our optimization model and technique, we use a set $P = \{p_1 \dots p_8\}$ of eight programmers and an example matrix dP shown in Fig. 1. The programmers' runtimes vary in the range 18.0 to 97.0 days (see matrix principal diagonal), and the overall runtime is 440 days. The runtime changes are in the range –19.2% to 18,6%.

$dP =$

	0	1	2	3	4	5	6	7
0	30.0	0.0	-6.7	-16.8	-2.7	-11.9	-11.7	-6.4
1	-11.9	18.0	-11.4	-15.6	-16.5	-8.3	9.2	1.7
2	-11.0	11.9	72.0	12.0	-7.9	-8.7	2.0	5.4
3	-6.6	-6.7	-8.9	35.0	18.6	8.2	6.9	-8.0
4	11.6	-6.2	3.8	16.5	89.0	5.1	-4.9	16.2
5	12.5	13.0	-10.2	0.0	-16.4	97.0	16.5	-2.0
6	5.3	-15.5	-2.1	10.3	13.4	10.2	58.0	0.0
7	-12.3	-19.2	3.7	10.4	-13.3	14.1	-12.8	41.0

Fig. 1. Example matrix *dP* of programmers' pairwise runtime changes in percent within one team

Fig. 2 depicts an example dT matrix that is computed over the *dP* (Fig. 1) using (1–3) and is a representation of team *single*.

$dT =$

	0	1	2	3	4	5	6	7
0	18.8	0.0	-4.8	-5.9	-2.4	-11.5	-6.8	-2.6
1	-3.6	13.8	-8.2	-5.5	-14.7	-8.1	5.3	0.7
2	-3.3	2.1	49.1	4.2	-7.0	-8.4	1.2	2.2
3	-2.0	-1.2	-6.4	40.8	16.5	8.0	4.0	-3.3
4	3.5	-1.1	2.7	5.8	66.9	5.0	-2.8	6.6
5	-3.7	2.3	-7.3	0.0	-14.6	105.6	9.6	-0.8
6	1.6	-2.8	-1.5	3.6	11.9	9.9	61.1	0.0
7	-3.7	-3.5	2.6	3.6	-11.8	13.7	-7.4	43.8

Fig. 2. Example matrix dT of programmers' runtimes (days) and pairwise changes of runtime in team *single*

The programmers' runtimes in the team vary in range –13.8 to 105.6 days, because the runtime changes vary in range –14.7 to 16.5 days. The overall runtime of team *single* calculated with (4) equals 399.9 days.

Fig. 3 shows an example dS matrix that is computed over the dT matrix. As many as 18 pairs of programmers included in the same team reduce the project runtime in range –0.3 down to –15.8 days each. The overall runtime can be decreased by –128.2 days. Each of rest 10 pairs increases the runtime in range 0.3 to 22.3 days. The overall runtime can be increased by 88.1 days. The sum of –128.2 + 88.1 = –40.1 days shows the overall runtime reduction in team *single*, which is –9.11 % against the overall runtime of all one-programmer teams. The vector of runtime changes of all programmers within team *single* is Δt^{single} = (–11.2, –4.2, –22.9, 5.8, –22.1, 8.6, 3.1, 2.8). The runtime of programmers 0, 1, 2 and 4 is decreased, therefore they have gained from establishing the team. The runtime of programmers 3, 5, 6 and 7 is increased, therefore they have lost when included in the team. As a result, the workload of each programmer in team *single* has been changed: t^{single} = (18.8, 13.8, 49.1, 40.8, 66.9, 105.6, 61.1, 43.8). In general, the team has gained in the runtime.

$dS =$

	0	1	2	3	4	5	6	7
0	18.8	-3.6	-8.1	-7.9	1.1	-15.2	-5.2	-6.3
1		13.8	-6.1	-6.7	-15.8	-5.8	2.5	-2.8
2			49.1	-2.2	-4.3	-15.7	-0.3	4.8
3				40.8	22.3	8.0	7.6	0.3
4					66.9	-9.6	9.1	-5.2
5						105.6	19.5	12.9
6							61.1	-7.4
7								43.8

Fig. 3. Matrix dS of pairwise total programmers' runtime changes (days) in team *single*

Matrix dT allows the evaluation of a runtime reduction potential of each programmer p_i included in team *single*. The change of the own runtime of the programmer is

$$dT_i^{own} = \sum_{j \neq i} dT_{j,i}. \quad (6)$$

The change of runtimes of all other programmers caused by programmer p_i is

$$dT_i^{oth} = \sum_{j \neq i} dT_{i,j}. \quad (7)$$

The overall change associated with programmer p_i is $dT_i^{all} = dT_i^{own} + dT_i^{oth}$. The lower value of dT_i^{all},

the higher potential p_i has with respect to reduction of the overall team runtime.

Table 1 reports the runtime changes each of the eight programmers can cause. It can be observed that programmer p_0 has the largest reducing potential of –45.2 days, and programmer p_6 has the largest increasing potential of 25.8 days. The programmers with the reducing potential must be included in a programming team first.

Table 1

Influence of compatibility of programmers on runtime changes

No	dT_i^{own}	dT_i^{oth}	dT_i^{all}	Priority
0	–11.2	–34.0	–45.2	1
1	–4.2	–34.1	–38.3	2
2	–22.9	–9.0	–31.9	3
3	5.8	15.6	21.4	7
4	–22.1	19.7	–2.4	6
5	8.6	–14.5	–5.9	4
6	3.1	22.7	25.8	8
7	2.8	–6.5	–3.7	5

Formulation of optimization problem. Let Ω be a set of all possible partitioning of set P of programmers into a set G of teams. Then we formulate the optimization problem as where T^G is defined by (5). Two cases are possible in solving (8): 1) $dT_{i,j}$, $i \neq j$, defined by (1) is constant; 2) $dT_{i,j}$ is variable. In the paper, we assume that $dT_{i,j}$ is constant. Since $dT_{i,j}$ can be both positive and negative, and the clique partitioning problem [12, 13] related to (8) is NP-hard, the problem (8) is also NP-hard. To solve it for large programmer sets, we propose a heuristic greedy algorithm of pairwise merge of teams that gives a maximum runtime reduction at each step.

$$\min_{G \in \Omega} = T^G \qquad (8)$$

Greedy algorithm of stepwise pairwise merge of teams (*GAMT*). It is described by Algorithm 1. Its inputs are the set P of programmers, vector t of their runtimes, and matrix dS of runtime changes. Its outputs are the set G of teams and the changed runtime $T(G)$ accounting for the programmer's compatibilities. The algorithm starts with n teams each consisting of a single programmer. The overall runtime of the teams is the sum of programmer's basic runtimes. At every iteration of the *while* loop, *GAMT* chooses a pair g' and g'' of teams whose merge gives a maximum ΔT^{best} of runtime reduction. The positive value of ΔT^{best} means that the runtime cannot be decreased, therefore variable *go* is set to *false* and the loop operation is over. If ΔT^{best} is negative, set G of teams is reconstructed: teams g' and g'' are merged into team g and are removed from G, then g is added to G. The teams' overall runtime $T(G)$ is reduced by ΔT^{best}. Since it is a maximal reduction, the algorithm is called greedy.

Algorithm 2 called *RuntimeChange* calculates the change RC of runtime when merging two teams g_j and g_k. The compatibility of each programmer v of team g_j with each programmer u of team g_k is accounted. The compatibility of all programmers within team g_j and within team g_k has been accounted for at previous iterations of the *while* loop.

Algorithm 1: Greedy algorithm of stepwise pairwise merge of teams (*GAMT*)

Input: A set $P = (p_0 \ldots p_{n-1})$ of programmers
Input: A vector t of programmer basic runtimes
Input: A matrix $dS[n \times n]$ of programmer runtime changes
Output: A set G of programming teams
Output: A runtime $T(G)$ of programming teams

```
G ← ∅   T(G) ← 0   go ← true
for i ← 0 to n − 1 do
    g_i ← {p_i}     T(g_i) ← t_i
    ΔT(g_i) ← 0
    G ← G ∪ {g_i}
    T(G) ← T(G) + t_i
while (go) do
    go ← false
    ΔT^best ← 0
    for j ← 0 to |G| − 1 do
        for k ← j + 1 to |G| − 1 do
            Δt_{j,k} ← RuntimeChange(P, dS, g_j, g_k)
            if ΔT^best > Δt_{j,k} then
                ΔT^best ← Δt_{j,k}   g' ← j   g'' ← k
    if ΔT^best < 0 then
        go ← true
        g ← g' ∪ g''
        G ← (G \ {g', g''}) ∪ {g}
        T(G) ← T(G) + ΔT^best
return G, T(G)
```

Algorithm 2: Calculation of runtime change after merging two teams (*RuntimeChange*)

Input: A set $P = (p_0 \ldots p_{n-1})$ of programmers
Input: A matrix $dS[n \times n]$ of programmer runtime changes
Input: two teams g_j and g_k of programmers which are candidates for merging
Output: A change RC of runtime of two teams merged

```
RC ← 0
for v ∈ g_j do
    for u ∈ g_k do
        if v < u then
            RC ← RC + dS_{v,u}
        else
            RC ← RC + dS_{u,v}
return RC
```

The *GAMT* algorithm applied to the example set of programmers has yielded 3 teams consisting of 5, 1 and 2 programmers: $G = \{\{0, 1, 2, 4, 5\}, \{3\}, \{6, 7\}\}$. Fig. 4 and Table 2 show that in first team, 9 pairs of programmers reduce the runtime by –83.1 days, and only one pair increases it by 1.1 days. Eight pairs of programmers from different teams have negative time changes of overall sum –36.6 days, which did not reduce the runtime. Nine pairs with positive time changes not included in the same team did not increase the runtime by 87.0 days.

$TT =$

	0	1	2	4	5	3	6	7
0	22.9	-3.6	-8.1	1.1	-15.2	-7.9	-5.2	-6.3
1		21.3	-6.1	-15.8	-5.8	-6.7	2.5	-2.8
2			54.4	-4.3	-15.7	-2.2	-0.3	4.8
4				50.3	-9.6	22.3	9.1	-5.2
5					74.0	8.0	19.5	12.9
3						35.0	7.6	0.3
6							50.6	-7.4
7								41.0

Fig. 4. Matrix *TT* of teams and their runtimes given by algorithm *GAMT* (rows and columns are reordered)

Table 2

Programming teams optimized by greedy algorithm *GAMT*

Team	1	2	3	All
Programmers	0, 1, 2, 4, 5	3	6, 7	0 – 7
Run-time, day	306	35	99	440
Reduction, day	–83.1	0	–7.4	–90.5
New runtime, day	222.9	35	91.6	349.5
Reduction, %	–27.2	0	–7.47	–20.6

Table 3 shows that the changes in runtimes of programmers of the *greedy* teams are $\Delta t^{greedy} = (-7.1, 3.3, -17.6, 0, -38.7, -23.0, -7.4, 0)$. The runtime of only one programmer has increased. The resulting runtimes of the programmers are $t^{greedy} = (22.9, 21.3, 54.4, 35, 50.3, 74, 50.6, 41)$.

Table 3

Comparison in runtime (days) of three allocations of programmers to teams

Progr. no.	Runtime	Team *single*		Teams *greedy*	
		Time	Change	Time	Change
0	30	18.8	–11.2	22.9	–7.1
1	18	13.8	–4.2	21.3	3.3
2	72	49.1	–22.9	54.4	–17.6
3	35	40.8	5.8	35.0	0.0
4	89	66.9	–22.1	50.3	–38.7
5	97	105.6	8.6	74.0	–23.0
6	58	61.1	3.1	50.6	–7.4
7	41	43.8	2.8	41.0	0.0
All	440	399.9	–40.1	349.5	–90.5

The *greedy* teams give the overall runtime reduction of –20.6% which is better against –9.11% of the *single* team. It should be noted that algorithm *GAMT* can be parallelized to handle large sets of programmers by using methods from [14].

Results. We have developed a software for the optimization of programming teams in the C++ language using Visual Studio 2022 under OS Windows 10. The experiments were done on Intel Core i7-10700 CPU processor on various sets *P* of programmers, vectors *t* of runtimes and matrices *dP* of runtime changes.

Fig. 5 compares on 10 runs of the software the overall runtimes of teams composed of 30 programmers that were generated by the *greedy GAMT* algorithm over the one-programmer teams and the team *single*. Since vector *t* and matrix *dP* were different on all 10 runs, the graphic represented in Fig. 5 using triangles and dash lines shows different properties of teams *single* which can increase the runtime up to 8.9% and decrease it down to –13.9% compared to the one-programmer teams. *GAMT* has generated and optimized 3 to 7 teams whose runtimes are up to –27.9% and up to –25.3% lower than the runtimes of corresponding one-programmer teams and team *single*.

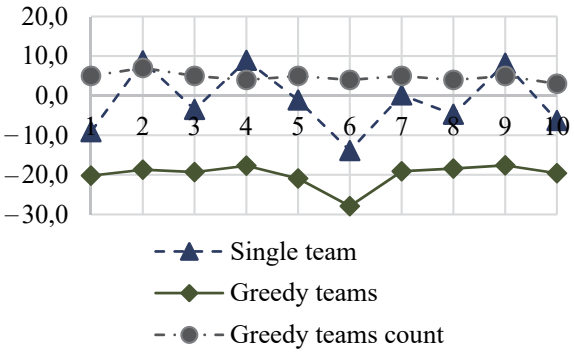

Fig. 5. Changes in runtime (%) of team *single* (triangles) and *greedy* teams (diamonds) over one-programmer teams, and *greedy* teams count (circles) on 30 programmers vs. 10 runs

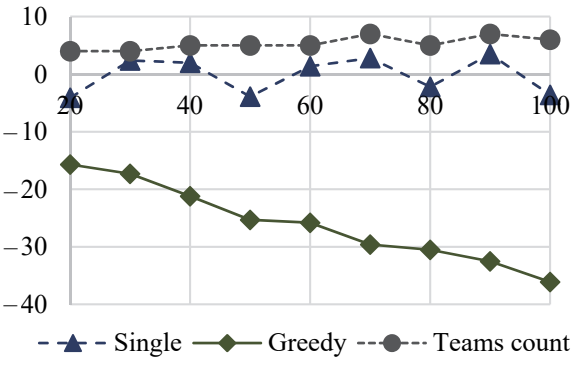

Fig. 6. Changes in runtime (%) of *single* team (triangles) and *greedy* teams (diamonds) against one-programmer teams, and *greedy* teams count (circles) vs. programmer count

Fig. 6 compares the overall runtimes of one-programmer teams, *single* teams, and *greedy* teams obtained by *GAMT* that are composed of 20 to 100 programmers. The vector *t* and matrix *dP* (average value of element is 5%) were unique for each set of programmers, therefore, the *single* team runtime differed from –4.1% to 3.5% compared to the one-programmer teams. The key pattern of *GAMT* is that the *greedy* teams' runtime is decreased over the growth of the set of programmers. While the number of programmers has been increased from 20 to 100, the *greedy* teams' runtime has been decreased from –15.7% down to –36.1%. The number of *greedy* teams has grown from 4 to 7.

Conclusion. The compatibility of programmers and their ability to efficiently work on a common IT project is one of the main sources of increasing the performances of programming teams and decreasing the teams' runtime. The paper has proposed to describe the compatibility with a matrix whose elements determine how one programmer can decrease or increase the runtime of another programmer when both are included in the same team. The matrix allows to find the number of teams, the size of each team and to establish the teams staff in such a way as to reach a maximum reduction of the overall runtime. The greedy algorithm of stepwise merge of teams that is developed in the paper and is implemented in the C++ language is heuristic and finds sub-optimal solutions. The conducted experiments have shown the runtime reduction is increased with the growth of size of the programmer set and is up to 36% for a set of one hundred programmers. They have also shown that the reduction depends on the properties of the compatibility matrix.

References

1. Masood Z., Hoda R., Blincoe K. Exploring Workflow Mechanisms and Task Allocation Strategies in Agile Software Teams. In: Baumeister H., Lichter H., Riebisch M. (eds). Agile Processes in Software Engineering and Extreme Programming. XP 2017. *Lecture Notes in Business Information Processing*, 2017, vol. 283. Springer, Cham. https://doi.org/10.1007/978-3-319-57633-6_19.

2. Britto R., Neto P. S., Rabelo R., Ayala W. and Soares T. A hybrid approach to solve the agile team allocation problem. *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8, DOI: 10.1109/CEC.2012.6252999.

3. Rachlin J. [et al.]. A-Teams: An Agent Architecture for Optimization and Decision-Support. In: Müller J. P., Rao A. S., Singh M. P. (eds). Intelligent Agents V: Agents Theories, Architectures, and Languages. ATAL, 1998. *Lecture Notes in Computer Science*, 1999, vol. 1555. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-49057-4_17.

4. Wrike. Available at: https://www.wrike.com/ (accessed: 29.03.2023).

5. Flow. Available at: https://www.getflow.com/ (accessed: 29.03.2023).

6. Gutierrez J. H., Astudillo C. A., Ballesteros-Perez P., Mora-Melia D. and Candia-Vejar A. The multiple team formation problem using sociometry. *Computers and Operations Research*, 2016, vol. 75, pp. 150–162. DOI: https://doi.org/10.1016/j.cor.2016.05.012.

7. Prihozhy A. A., Zhdanouski A. M. Method of qualification estimation and optimization of professional teams of programmers. *Sistemnyy analiz i prikladnaya informatika* [System analysis and applied information science], 2018, no. 2, pp. 4–11. https://doi.org/10.21122/2309-4923-2018-2-4-11 (In Russian).

8. Prihozhy A. A. Exact and greedy algorithms of allocating experts to maximum set of programmer teams. *System analysis and applied information science*, 2022, no. 1, pp. 40–46. https://doi.org/10.21122/2309-4923-2022-1-40-46.

9. Prihozhy A., Zhdanouski A. Genetic algorithm of optimizing the size, staff and number of professional teams of programmers. *Open Semantic Technologies for Intelligent Systems*, Minsk, BSUIR Publ., 2019, pp. 305–310.

10. Prihozhy A. A., Zhdanouski A. M. Genetic algorithm of optimizing the qualification of programmer teams. *System analysis and applied information science*, 2020, no. 4, pp. 31–38. https://doi.org/10.21122/2309-4923-2020-4-31-38.

11. Prihozhy A. A., Zhdanouski A. M. Genetic algorithm of allocating programmers to groups. *Nauka – obrazovaniyu, proizvodstvu, ekonomike: materialy 13-y Mezhdunarodnoy nauchno-prakticheskoy konferentsii* [Science to education, industry and economics: Proceedings of 13th international scientific and practical conference]. Minsk, 2015, vol. 1, pp. 286–287 (In Russian).

12. Grotschel M., Wakabayashi Y. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 1989, vol. 45, no. 1, pp. 59–96. https://doi.org/10.1007/BF01589097.

13. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms. *System analysis and applied information science*, 2021, no. 3, pp. 40–50.

14. Prihozhy A. A. Analysis, transformation and optimization for high performance parallel computing. Minsk, BNTU Publ., 2019. 229 p.

Список литературы

1. Masood Z., Hoda R., Blincoe K. Exploring Workflow Mechanisms and Task Allocation Strategies in Agile Software Teams // Agile Processes in Software Engineering and Extreme Programming H. Baumeister,

H. Lichter, M. Riebisch (eds). XP 2017 [et. al.] // Lecture Notes in Business Information Processing. 2017. Vol. 283. Springer, Cham. https://doi.org/10.1007/978-3-319-57633-6_19.

2. A hybrid approach to solve the agile team allocation problem / R. Britto [et. al.] // 2012 IEEE Congress on Evolutionary Computation. 2012. P. 1–8. https://doi: 10.1109/CEC.2012.6252999.

3. A-Teams: An Agent Architecture for Optimization and Decision-Support / J. Rachlin [et al.] // Intelligent Agents V: Agents Theories, Architectures, and Languages, ATAL 1998 / J. P. Müller, A. S. Rao, M. P. Singh (eds) // Lecture Notes in Computer Science. 1999. Vol. 1555. Springer. 1999. https://doi.org/10.1007/3-540-49057-4_17.

4. Wrike. URL: https://www.wrike.com/ (дата обращения: 29.03.2023).

5. Flow. URL: https://www. getflow.com/ (дата обращения: 29.03.2023).

6. The multiple team formation problem using sociometry / Gutierrez J. H. [et al.] // Computers and Operations Research. 2016. Vol. 75. P. 150–162. https://doi.org/10.1016/j.cor.2016.05.012.

7. Прихожий А. А., Ждановский А. М. Метод оценки квалификации и оптимизация состава профессиональных групп программистов // Системный анализ и прикладная информатика. 2018. №. 2. С. 4–11. https://doi.org/10.21122/2309-4923-2018-2-4-11.

8. Prihozhy A. A. Exact and greedy algorithms of allocating experts to maximum set of programmer teams // System analysis and applied information science. 2022, no. 1, pp. 40–46. https://doi.org/10.21122/2309-4923-2022-1-40-46.

9. Prihozhy A., Zhdanouski A. Genetic algorithm of optimizing the size, staff and number of professional teams of programmers // Open Semantic Technologies for Intelligent Systems, Minsk, BSUIR, 2019, pp. 305–310.

10. Prihozhy A. A., Zhdanouski A. M. Genetic algorithm of optimizing the qualification of programmer teams // System analysis and applied information science. 2020, no. 4, pp. 31–38. https://doi.org/10.21122/2309-4923-2020-4-31-38.

11. Прихожий А. А., Ждановский А. М. Генетический алгоритм разбиения коллектива программистов на группы // Наука – образованию, производству, экономике: материалы 13-й Междунар. науч.-практ. конф. Минск: БНТУ, 2015. Т. 1. С. 286–287.

12. Grotschel M., Wakabayashi Y. A cutting plane algorithm for a clustering problem // *Mathematical Programming*. 1989. Vol. 45, no. 1. P. 59–96. https://doi.org/10.1007/BF01589097.

13. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms // *System analysis and applied information science*. 2021, no. 3, pp. 40–50.

14. Prihozhy A. A. Analysis, transformation and optimization for high performance parallel computing. Minsk: BNTU, 2019. 229 p.

Information about the author

Prihozhy Anatoly Alekseevich − DSc (Engineering), Professor, Professor, the Department of Computer and System Software. Belarusian National Technical University (65, Nezalezhnasti Ave., 220013, Minsk, Republic of Belarus). E-mail: prihozhy@yahoo.com

Информация об авторе

Прихожий Анатолий Алексеевич − доктор технических наук, профессор, профессор кафедры программного обеспечения информационных систем и технологий. Белорусский национальный технический университет (220013, г. Минск, пр. Независимости, 65, Республика Беларусь). E-mail: prihozhy@yahoo.com

Received 15.04.2023