UDC 004.4

RANDOM ALGORITHM OF FORMING PROGRAMMING TEAMS ACCOUNTING FOR COMPATIBILITY OF PROGRAMMERS

Prihozhy A. A.
Belarusian National Technical University,
Minsk, Belarus, prihozhy@yahoo.com

Abstract. The compatibility of programmers is one of the main sources of increasing the efficiency of operation of programming teams. In the paper, we have proposed a random algorithm of forming teams which account for compatibility of programmers and their influence on the teams' runtime costs. Experimental results have shown that the random algorithm forms the teams which reduce the runtime costs by up to 12,49 % compared to the teams generated by the greedy algorithm.

Key words: algorithm, random algorithm, RGAMT, programming teams, agile.

Formal methods of forming programming teams have not received too much attention from scientists. Agile [1; 2] is a set of values and principles of developing software over joint efforts of development teams and customers. Paper [3] emphasizes that a successful software development team must be made up of competent developers. It presents a hybrid approach based on the *NSGA*-II multi-objective metaheuristic and Mamdani Fuzzy Inference Systems to solve the agile team allocation problem. The agile team formation is a NP-hard problem. Agent-based evolutionary methods of optimization [4] aim at performing the management of teams. Papers [5–7] propose tools that increase the productivity and efficiency of teams working on various projects. Paper [8] proposes a method of formalizing the level of teams' competency, and paper [9] solves the problem of allocating experts to maximum set of programming teams. Papers [10–13] developed a genetic algorithm-based approach for forming the teams. This paper proposes a random algorithm of forming teams which accounts for compatibility of programmers and their influence on runtime costs.

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 programmers' runtime costs to be spent while working on the project. Let $G = \{g_1 \ldots g_k\}$ be a set of teams the programmers must be allocated to. The runtime of programmers included in the same team should be corrected accounting for the programmers' compatibility. Element (i, j) of input matrix dP is a positive / negative change of programmer j's runtime t_j in percent caused by programmer i. Matrix dT calculated through dP represents pairwise changes of the programmers' runtimes due to including the programmers in the same team. If programmer j is included in team g, the runtime t_j is changed to $t_j(g)$ that is evaluated as

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

where

$$dT_j(g) = \sum_{i \in g, i \neq j} \left(dP_{i,j} / 100 \right), \tag{2}$$

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

$$T(g) = \sum_{i \in g} t_j(g) \tag{3}$$

and the overall runtime of programming teams of set G is

$$T^G = \sum_{g \in G} T(g). \tag{4}$$

The compatibility of each programmer with all other programmers is evaluated because of forming a team *single* = *P*. Let Ω be a set of all possible partitioning of set *P* of programmers. An element of set Ω is a set *G* of teams established on the set *P* of programmers. The optimization problem we solve in the paper is

$$\min_{G \in \Omega} \{T^G\}. \tag{5}$$

Like work [13], this paper assumes that the elements of matrices *dP* and *dT* remain the same after adding a programmer to a team during solving problem (5). Since elements $dT_{i,j}$ can be both positive and negative, problem (5) is similar to the clique partitioning problem [14; 15] which is NP-hard. To solve (5) for large sets of programmers, a greedy algorithm of the stepwise pairwise merge of teams is proposed in [13]. In this paper we extend the algorithm to a random one with the aim of reducing the runtime of programmers in teams.

Algorithm 1 describes the random greedy stepwise pairwise merge of teams (*RGAMT*). Its inputs are the set *P* of programmers, vector *t* of programmers' runtimes, and matrix *dT* of runtime changes. Its outputs are the set *G* of teams and the changed runtime costs *T*(*G*) which accounts for the programmer's compatibilities. The algorithm starts with *n* teams each consisting of a single programmer. In the *while* loop, it determines for each team another best team for merging (array *BestC*), which yields a maximum of the runtime costs reduction.

Algorithm 1: Random greedy algorithm of stepwise pairwise merge of teams (*RGAMT*)

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

$G \leftarrow \varnothing$ $T(G) \leftarrow 0$ $go \leftarrow true$
for $i \leftarrow 0$ **to** $n - 1$ **do**
  $g_i \leftarrow \{p_i\}$ $T(g_i) \leftarrow t_i$
  $G \leftarrow G \cup \{g_i\}$
  $T(G) \leftarrow T(G) + t_i$
while (*go*) **do**

```
go ← false
for j ← 0 to |G| – 1 do
    BestC(g_j).ΔT ← ∞
    BestC(g_j).team ← undefined
    for k ← 0 to |G| – 1 do
        if j ≠ k then
            ΔT(g_j, g_k) ← RuntimeChange(t, dT, g_j, g_k)
            if BestC(g_j).ΔT > ΔT(g_j, g_k) then
                BestC(g_j).ΔT ← ΔT(g_j, g_k)
                BestC(g_j).team ← g_k
g' ←SelectPairRandomly(G, BestC)
g'' ← BestC(g').team
if BestC(g').ΔT < 0 then
    go ← true
    g ← g' ∪ g''
        T(g) ← T(g') + T(g'') + BestC(g').ΔT
    G ← (G \ {g', g''}) ∪ {g}
    T(G) ← T(G) + BestC(g').ΔT
return G, T(G)
```

Algorithm *RuntimeChange* calculates the overall reduction (if possible) of runtimes of potentially merged teams g_j and g_k using (2) for estimating the influence of each programmer of team g_j on the runtime of each programmer of team g_k, and wise versa. Vector t and matrix dT are used for the calculation. Algorithm 2, *SelectPairRandomly* selects the elements of *BestC* with negative values of ΔT and randomly chooses one of them using the roulette rule. The teams g' and g'' with smaller negative value of ΔT are assigned a larger probability and have bigger chances to be chosen for merging. Teams g' and g'' are removed from G and a new team $g' \cup g''$ is added to G. If for each element of *BestC* the value of ΔT is positive the merging process is over.

Algorithm 2: Random selection of teams to be merged (*SelectPairRandomly*)

Input: A set G of programming teams
Input: A vector *BestC* associating each team of G with other team yielding minimum runtime after merging
Output: A team $g' \in G$ randomly selected for merging
Output: A team $g'' = BestC(g').team$ to be merged with g'

```
G* ← ∅      ΔT ← 0
for each q ∈ G do
    if BestC(q).ΔT < 0 then
        G* ← G* ∪ {q}
        ΔT ← ΔT – BestC(q).ΔT
for each q ∈ G* do
    probability(q) ← – BestC(q).ΔT / ΔT
rnd ← random()      g' ← undefined
for each q ∈ G* do
```

if *rnd* ≤ *probability*(*q*) **then**
    g' ← *q*
    break for
else
    rnd ← *rnd* – *probability*(*q*)
return *g'*

Algorithm 3 organizes the multiple execution of Algorithm 1 (*RGAMT*) and the selection of partitioning *G* which provides the smallest overall runtime of programmers assigned to the teams of *G* along all iterations of the loop. The control parameter *Iter* defines the number of attempts to find a better solution G^{best} and to improve the runtime $T(G^{best})$.

If large sets of programmers are assigned to programming teams, the optimization algorithms can consume much CPU time. In this case, the parallelization approach is attractive [16–19], which can speed up computations significantly on multi-core processors and multi-processor systems.

Results. We have developed a software written in the C++ language under Visual Studio 2022 and OS Windows 10 for forming and optimizing programming teams accounting for the compatibility of programmers. 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 programmers' runtimes changes. Figure 1 shows that forming the team single can both increase and decrease the overall runtime of programmers against the teams of separate programmers. The greedy algorithm *GAMT* has decreased the overall teams' runtime by 7,76–34,2 %, while the random algorithm *CRGA-RGAMT* has decreased it by 9,93–36,86 %. Figure 2 shows that the random algorithm *CRGA-RGAMT* has outperformed the greedy algorithm *GAMT* by 0,11–4,03 % and by 4,55–12,49 % at ±5 % and ±10 % of average value of the matrix *dP*'s element respectively.

Algorithm 3: Control of multiple execution of *RGAMT* algorithm (*CRGA*)

Input: A set $P = (p_0 \ldots p_{n-1})$ of programmers
Input: A vector *t* of programmer basic runtimes
Input: A matrix *dT* of programmer runtime changes
Input: A number *Iter* of *RGAMT* execution
Output: A best set G^{best} of programming teams
Output: A smallest runtime $T(G^{best})$ of partitioning G^{best}
G^{best} ← *undefined* $T(G^{best}) \leftarrow \infty$
for *i* ← 1 **to** *Iter* **do**
    G ← *RGAMT*(*P*, *t*, *dT*)
if $T(G^{best}) > T(G)$ **then**
    $T(G^{best}) \leftarrow T(G)$ $G^{best} \leftarrow G$
return G^{best}, $T(G^{best})$

Conclusion.

The compatibility of programmers is one of the main sources of increasing the efficiency of operation of programming teams. In the paper, we have proposed a random algorithm of forming teams which accounts for compatibility of programmers working on the same IT project. The algorithm is based on the recently proposed greedy algorithm of pairwise stepwise merging of teams initialized by including each programmer in a separate team. It is shown by conducting computational experiments and analyzing the obtained results that the random algorithm forms the teams which reduce the runtime costs of programmers working on the IT project compared to the teams generated by the greedy algorithm. The random algorithm can be easily parallelized to exploit resources of multi-core processors and multiprocessor systems.

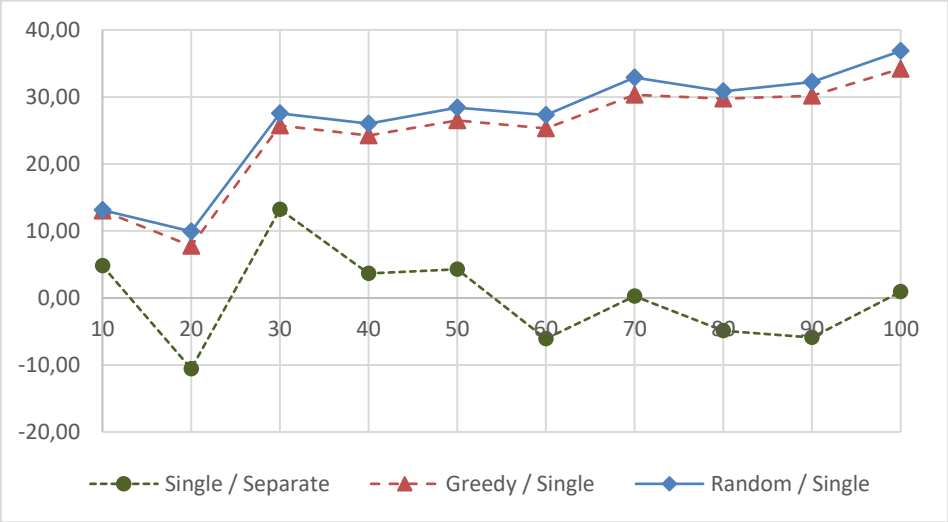

Figure 1 – Comparison (%) of separate, single, greedy (algorithm *GAMT*) and random (algorithm *CRGA-RGAMT*) teams regarding the overall runtime vs. number of programmers

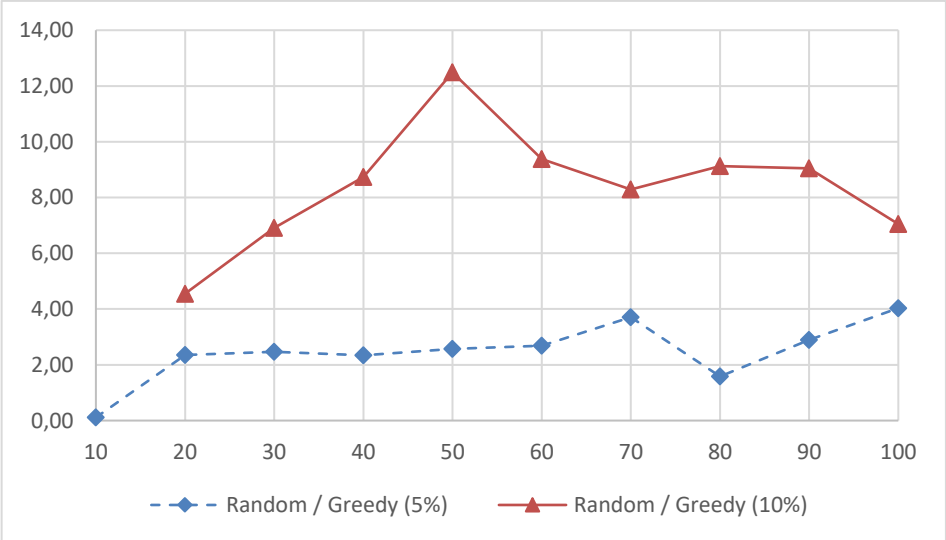

Figure 2 – Reduction (%) of overall teams' runtime costs of random *CRGA-RGAMT* algorithm against greedy *GAMT* algorithm vs. number of programmers at ±5 % (rectangular) and ±10 % (triangular) of values of matrix *dP*'s elements on average

References:

1. Joshi, S. Agile Development – Working with Agile in a Distributed Team Environment / S. Joshi // MSDN Magazine, – 2012. – Vol. 27, No. 1. – P. 1–6.
2. 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.
3. A hybrid approach to solve the agile team allocation problem / R. Britto [et al.] // 2012 IEEE Congress on Evolutionary Computation. – 2012. – P. 1–8.
4. 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.
5. Wrike [Electronic resource]. – Mode of access: https://www.wrike.com/. – Date of access: 10.11.2023.
6. Flow [Electronic resource]. – Mode of access: https://www.getflow.com/. – Date of access: 10.11.2023.
7. The multiple team formation problem using sociometry / J. H. Gutierrez [et al.] // Computers and Operations Research. – 2016. – Vol. 75. – P. 150–162.
8. Прихожий, А. А. Метод оценки квалификации и оптимизация состава профессиональных групп программистов / А. А. Прихожий, А. М. Ждановский // Системный анализ и прикладная информатика. – 2018. – № 2. – С. 4−11.
9. Prihozhy, A. A. Exact and greedy algorithms of allocating experts to maximum set of programmer teams / A. A. Prihozhy // System analysis and applied information science. – 2022. – № 1. – P. 40–46.
10. Prihozhy, A. Genetic algorithm of optimizing the size, staff and number of professional teams of programmers / A. Prihozhy, A. Zhdanouski // Open Semantic Technologies for Intelligent Systems – Minsk : BSUIR Publ., 2019. – P. 305–310.
11. Prihozhy, A. A. Genetic algorithm of optimizing the qualification of programmer teams. / A. A. Prihozhy, A. M. Zhdanouski // System analysis and applied information science. – 2020. – № 4. – P. 31–38.
12. Prihozhy, A. A., Zhdanouski, A. M. Genetic algorithm of allocating programmers to groups / A. A. Prihozhy, A. M. Zhdanouski // Science to education, industry and economics: Proceedings of 13th international conference. – Minsk : BNTU Publ., 2015. – Vol. 1. – P. 286–287.
13. Prihozhy, A. A. Optimization of programming teams on compatibility of programmers / A. A. Prihozhy // Proceedings of BSTU, issue 3, Physics and Mathematics. Informatics. – 2023. – No. 2 (272). – P. 104–110.
14. Grotschel, M. A cutting plane algorithm for a clustering problem / M. Grotschel, Y. Wakabayashi // Mathematical Programming. – 1989. – Vol. 45, No. 1. – P. 59–96.
15. Prihozhy, A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms / A. A. Prihozhy // System analysis and applied information science, 2021. – No. 3. – P. 40–50.

16. Prihozhy, A. A. Analysis, transformation and optimization for high performance parallel computing / A. A. Prihozhy. – Minsk : BNTU Publ., 2019. – 229 p.

17. Prihozhy, A. A. Asynchronous scheduling and allocation / A. A. Prihozhy // Proceedings Design, Automation and Test in Europe. Paris, France. – IEEE, 1998. – P. 963–964.

18. Techniques for optimization of net algorithms / A. Prihozhy [et al.] // 2002 International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002), Warsaw, Poland, 22–25 September 2002. – IEEE, 2002. – P. 211–216.

19. Pipeline synthesis and optimization from branched feedback dataflow programs/ A. A. Prihozhy [et al.] // Journal of Signal Processing Systems, Springer Nature. – 2020. – Vol. 92. – P. 1091–1099.