UDC 004.4-004.9

PRIHOZHY A.A.

EXACT AND GREEDY ALGORITHMS OF ALLOCATING EXPERTS TO MAXIMUM SET OF PROGRAMMER TEAMS

Belarusian National Technical University

The allocation of experts to programmer teams, which meet constraints on professional competences relat-ed to programming technologies, languages and tools an IT project specifies is a hard combinatorial problem. This paper solves the problem of forming the maximum number of teams whose experts meet all the constraints within each team. It develops and compares two algorithms: a heuristic greedy and exact optimal. The greedy algorithm iteratively solves the set cover problem on a matrix of expert competences until can create the next workable team of remaining experts. The paper proves that the allocation greedy algorithm is not accurate even if the set cover algorithm is exact. We call the allo-cation algorithm as double greedy if the set cover algorithm is greedy. The exact algorithm we propose finds optimal solution in three steps: generating a set of all non-redundant teams, producing a graph of team's independency, and searching for a maximum clique in the graph. The algorithm of generating the non-redundant teams traverses a search tree con-structed in such a way as to guarantee the creation of all non-redundant teams and absorbing all redundant teams. The edges of the non-redundant team independency graph connect teams that have no common expert. The maximum clique search algorithm we propose accounts for the problem and graph features. Experimental results show that the exact algo-rithm is a reference one, and the double-greedy algorithm is very fast and can yield suboptimal solutions for large-size allocation problems.

Keywords: *programmer, team, competence, expert, allocation problem, optimization.*

Introduction

In the rapidly developing information technology industries, there is need to assemble teams of growing complexity to tackle problems on a larger scale than ever before. Agile is a set of values and principles of developing software and finding solutions over joint efforts of development teams and customers [1, 2]. Agent-based evolutionary optimization methods [3] aim at performing the management of teams.

The process of allocating tasks to teams has not received much attention. In [4], the authors describe the process of task allocation as including three mechanisms of workflow across teams and five types of task allocation strategies. In [5], the authors emphasize that a successful software development team has to be made up of competent developers. Competency is the ability of a developer to perform a job properly. It is a combination of knowledge, skills and attitudes used to improve performance. In [6-8], the authors proposed platforms that increase team's productivity and efficiency for various tasks and projects. In [9], a method for formalizing and evaluating the competency of individual programmers and entire programmer teams was proposed. Since the programmer allocation problem is combinatorial, the goal of works [10 - 12] was to develop a genetic-algorithm-based meta-heuristic approach for finding acceptable solutions of large-size problems at different requirements to competences of programmers.

In the paper, we formulate a combinatorial problem and propose a heuristic greedy and an exact optimal algorithm of allocating experts to a maximum set of programmer teams, assuming that two teams may not share the same expert. The contribution of the paper is as follows:

1. An algorithm of generating feasible non-redundant teams of experts is proposed;
2. A graph of non-redundant teams independency is introduced; the experts allocation problem is solved by searching for a maximum clique in the graph;
3. The experimental results obtained show that the heuristic greedy algorithm is very fast and gives good enough solutions against the exact algorithm.

Combinatorial problem formulation

Let $C = \{c_1,\ldots, c_m\}$ be a set of competences Joseph Sijin proposed in [13] in order to create the programmer competency matrix and to estimate the qualification of candidates to IT projects. He introduced four predefined competency levels $L0$, $L1$, $L2$ and $L3$, and formulated requirements for each of them regarding all the competences.

Let $P = \{p_1,\ldots p_n\}$ be a set of programmers who desire to work on an IT project and have evaluated the competency level on each of the topics. Table 1 describes a sample of 12 programmers characterized by 12 competences. It indicates the competency $Level(p, c)$ of each programmer p for each competence c.

Usually, each IT project establishes a constraint $Level(p, c) \geq \lambda_c$ for the level of each competence $c \in C$ at least one programmer p of the team must have. We also use notation λ for the overall competence: $\lambda_c = \lambda$ for all $c \in C$.

Table 1. Competence level of twelve programmers (case study)

Programmer	Competence											
	0	1	2	3	4	5	6	7	8	9	10	11
0	3	0	3	3	3	2	2	3	0	3	0	2
1	3	1	2	0	1	1	3	2	3	3	0	3
2	3	2	3	3	0	1	1	0	0	2	3	3
3	1	2	1	0	2	3	0	2	0	3	3	2
4	1	3	1	3	3	3	0	2	1	3	0	2
5	3	3	3	2	0	1	3	1	3	2	1	1
6	1	1	0	2	2	3	3	2	3	1	1	2
7	0	3	2	0	0	2	2	1	2	1	2	2
8	1	3	2	0	3	0	2	0	2	0	1	0
9	1	1	0	2	0	1	3	0	1	3	2	2
10	0	2	1	0	0	2	3	0	1	2	1	0
11	1	3	0	0	0	0	2	1	1	3	3	0

Table 2. Matrix Δ of expert competences at $\lambda \geq L2$ (case study)

Expert	Competence											
	0	1	2	3	4	5	6	7	8	9	10	11
0	+		+	+	+	+	+	+		+		+
1	+		+				+	+	+	+		+
2	+	+	+	+						+	+	+
3		+			+	+		+		+	+	+
4		+		+	+	+		+		+		+
5	+	+	+	+			+		+	+		
6				+	+	+	+	+	+			+
7		+	+			+	+		+		+	+
8		+	+		+		+		+			
9				+			+			+	+	+
10		+				+	+			+		
11		+					+			+	+	

We qualify a programmer who meets the constraint on at least one competence as expert. The working team must have an expert for each competence. Applying the constraint of $\lambda = L2$ to Table 1 generates Table 2, which describes a matrix $\Delta[n\times m]$ of expert competences. Symbol '+' indicates competences the experts have.

Definition 1. A team t is a subset of programmers $t \subseteq P$ such

that

$$\bigcup_{p \in t} \Delta_p = C \quad (1)$$

Definition 2. A team s is redundant if at least one programmer $r \in s$ exists such that team $t = s \setminus \{r\}$ meets (1).

Definition 3. A team t is non-redundant if for any programmer $r \in t$ inequality (2) holds.

$$\bigcup_{p \in t \setminus \{r\}} \Delta_p \neq C \quad (2)$$

Definition 4. A team t absorbs team e if $t \subset e$.

Definition 5. Teams t_i and t_j are independent if $t_i \cap t_j = \varnothing$.

Let Ω be a set of feasible allocations of experts of P to a set T of workable teams, assuming that size $|T|$ is not defined in advance. Our objective is to solve the following combinatorial problem:

$$\max_{T \in \Omega} |T| \quad (3)$$

subject to

$$\bigcup_{p \in t_i} \Delta_p = C \quad for\ all\ t_i \in T \quad (4)$$

Δ_p is a set of competences of expert p. Equation (5) estimates an upper bound of the team count.

$$upper(|T|) = \min_{c \in C} \left[\sum_{p \in P} \delta_{pc} \right] \quad (5)$$

where $\delta_{pc} = 1$ if Δ_{pc} = '+'. According to Table 2, $upper(|T|) = 4$.

Greedy algorithm of solving the problem

The greedy Algorithm 1 we propose heuristical-ly allocates experts to teams and finds a subopti-mal solution in general case. The algorithm itera-tively solve the well-known set cover problem [14], which is NP-complete, until the next worka-ble *team* cannot be created of the remaining ex-perts. Initially set R consists of all experts of set P, and set T of teams is empty. Each iteration of the loop forms a *team* of minimum size, which covers all competences, by solving the set cover problem. Then it removes experts of *team* from R and add the team to T. If *team* is empty, the algorithm ter-minates its operation.

Algorithm 1: Greedy allocation of experts to teams

Input: A set P of experts
Input: A set C of competences
Input: A matrix Δ of expert competences
Output: A set T of workable teams represented by subsets of experts

```
R ← P    T ← ∅    next_team ← true
while next_team do
        team ← SetCoverProblem (C, R, Δ)
        if team = ∅ then
                next_team ← false
        else
                T ← T ∪ { team }    R ← R \ team
return T
```

Algorithm 1 does not guarantee obtaining the accurate solution. Table 3 describes matrix Δ, which proves the assertion. Figure 1a shows three non-redundant teams that can be generated from Δ. Algorithm 1 selects team t_0 at the first iteration and returns $T = \{t_0\}$ after the second iteration. Figure 1b shows that the maximum-size solution is $T = \{t_1, t_2\}$, which represents a maximum clique of a team independency graph G_D. As Algorithm 1 is a heuristic one, it is reasonable to solve the set cover problem by the greedy algorithm [15]. In this case, Algorithm 1 becomes the double-greedy heuristic algorithm.

Table 3. The matrix Δ proves that greedy algorithm can give suboptimal solution

Expert	Competence				
	c_0	c_1	c_2	c_3	c_4
p_0	+	+	+		
p_1	+			+	+
p_2				+	
p_3					+
p_4			+		
p_5		+			

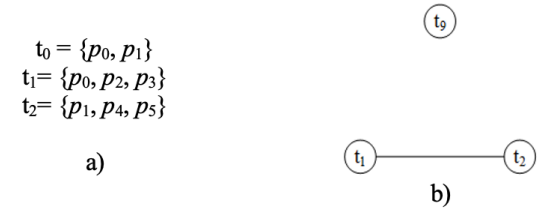

Figure 1 – Non-redundant teams of experts from Table 3: a) team members; b) team independency graph G_D

Generation of feasible non-redundant teams

A team search tree depicted in Figure 2 is a directed labeled acyclic graph supporting the generation of redundant and all non-redundant workable teams. All nonterminal vertices (without fill) of the tree correspond to programmers. There are four types of terminal vertex: a redundant workable team (in red); a non-redundant team (in green); a non-workable team which does not cover all competencies of C (in black); and a tree's branch represented as single vertex (in grey). There are two types of edge in the tree: on-edge (right outgoing solid line) and off-edge (left outgoing dash line). A path from root to a leaf t_k determines the team members. If the path includes an outgoing on-edge of vertex p_i then $p_i \in t_k$, if it includes an off-edge then $p_i \in t_k$.

Algorithm 1 does not guarantee obtaining the accurate solution. Table 3 describes matrix Δ, which proves the assertion. Figure 1a shows three non-redundant teams that can be generated from Δ. Algorithm 1 selects team t_0 at the first iteration and returns $T = \{t_0\}$ after the second iteration. Figure 1b shows that the maximum-size solution is $T = \{t_1, t_2\}$, which represents a maximum clique of a team independency graph G_D. As Algorithm 1 is a heuristic one, it is reasonable to solve the set cover problem by the greedy algorithm [15]. In this case, Algorithm 1 becomes the double-greedy heuristic algorithm.

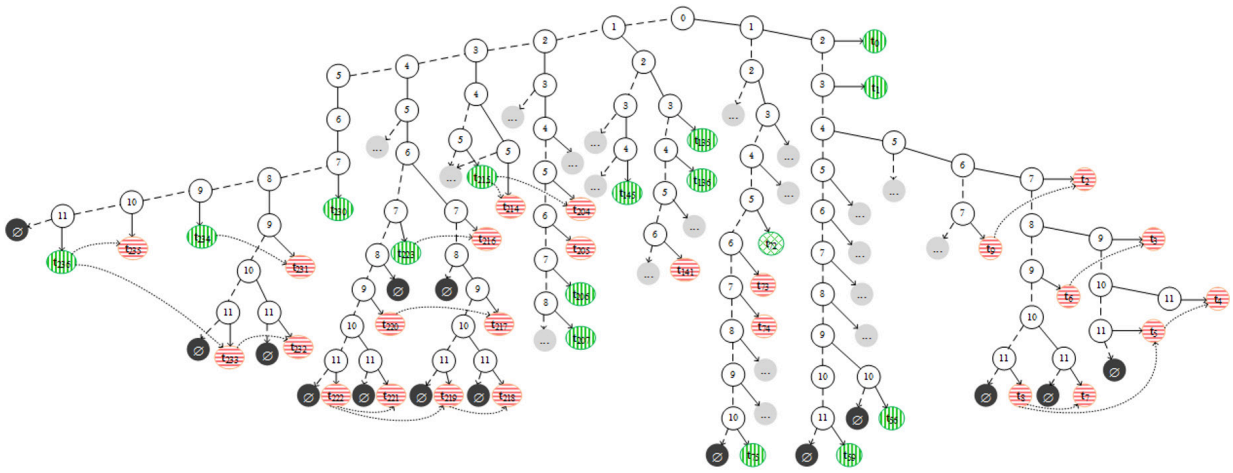

Figure 2 – Non-redundant teams search tree

Algorithm 2 uses four operation modes: *FORWARD, BACKWARD, SUCCESS*, and *FAILURE*. In mode *FORWARD*, it switches to mode *SUCCESS* if the programmers collected in stack *SS* have set *C* of competences. If the depth of stack *SM* is equal to n, the mode switches to *FAILURE*. Otherwise, the algorithm keeps the mode, pushes the topm programmer in stack *SS* adding programmer's competences to the current team, and passes from programmer topm to programmer topm + 1 through on-edge. In mode *SUCCESS*, the algorithm gener-ates new team, possibly absorbs the previously created teams of set *T*, adds the new team to *T*, and switches to mode *BACKWARD*. In *BACKWARD*, the algorithm performs backtracking while *SM* has off-edge at top. If the depth is equal to 0, the algo-rithm terminates operation returning T. If a record with on-edge found, the algorithm replaces it with off-edge and switches to the *FORWARD* mode.

Algorithm 2: Generation of non-redundant teams

Input: A set $P = \{0,\ldots, n\text{-}1\}$ of experts
Input: A set C of competences
Input: A matrix Δ of expert competences
Output: A set T of non-redundant teams represented by expert subsets

```
SM(0) ← true   SS(0).pr ← 0   SS(0).cs ← Δ(0)
mode ← FORWARD
go ← true   topm ← 1   tops ← 1   T ← ∅
while go do
    if mode = FORWARD then
        if SS(tops - 1).cs = C then
            mode ← SUCCESS
        else
            if topm = n then
                mode ← FAILURE
            else
                SS(tops).pr ← topm
                SS(tops).cs ← SS(tops - 1).cs ∪ Δ(topm)
                tops ← tops + 1
                SM(topm) ← true   topm ← topm + 1
    else if mode = SUCCESS then
        team ← ∅
        for i ← 1 to tops do  team ← team ∪ SS(i).pr
        T ← Absorption (team, T)
        T ← T ∪ {team}
        mode ← BACKWARD
    else if mode = BACKWARD then
        if SM(topm - 1) then
            SM(topm - 1) ← false   tops ← tops - 1
            mode ← FORWARD
        else
            if topm > 0 then topm ← topm – 1 else go ← false
    else if mode = FAILURE then
        while topm > 0 and SM(topm - 1) do
            SM(topm - 1) ← false
            topm ← topm – 1   tops ← tops – 1
        if tops = 0 then
            go ← false
        else
            topm ← SS(tops).pr + 1
            SM(topm - 1) ← false   tops ← tops – 1
            if tops = 0 then
                SS(tops).pr ← topm
                SS(tops).cs ← Δ(topm)   tops ← tops + 1
                SM(topm) ← true   topm ← topm + 1
            mode ← FORWARD
return T
```

In mode *FORWARD*, the algorithm switches to *FAILURE*. if it has generated an unworkable team passing through on-edges. In mode *FAILURE*, it performs backtracking over on-edges using both stacks to find a vertex, which allows the traversal of alternative paths in the search tree and allows the generation of alternative teams in the *FORWARD* mode.

The search tree generated by Algorithm 2 is de-picted in Figure 2. Totally, the tree includes 237 terminal team-vertices that represent 190 redun-dant (in red) and 47 non-redundant (in green) teams. The figure shows only part of generated branches, grey leafs represent tree branches con-taining other teams.

A path from tree root to team-leaf determines the team members. For instance, the path to t_9 includes nonterminal vertices 0,…,7. Vertices 0, 1, 4, 5 and 7 have outgoing on-edges (solid line). Vertices 2, 3 and 6 have outgoing off-edges (dash line). Therefore, $t_9 = \{p_0, p_1, p_4, p_5, p_7\}$.

In the search tree, dot-line edges show absorbing one team by other team. For example, team t_9 has outgoing dot-line edge pointing to team $t_2 = \{p_0, p_1, p_4, p_5, p_6, p_7\}$. Therefore, t_9 absorbs t_2 because $t_9 \subset t_2$.

The search tree has properties as follows:

1. In any path from root to leaf, the competences of predecessors does not include all competences of successors.

2. The competences of successors may completely include the competences of a predecessor.

3. As a result, a team may only absorb other redundant team located to right in the search tree.

4. Algorithm 2 finds all non-redundant teams for the given set of programmers and absorbs all redundant teams.

Figure 3 depicts a set of 47 non-redundant teams Algorithm 2

has generated over the tree from Figure 2. The rows correspond to teams, and the columns correspond to programmers. Value 1 indicates including a pro-grammer in a team.

Exact algorithm based on non-redundant team independency graph

In the undirected non-redundant team independency graph $G_D = (T, D)$, T is a set of non-redundant teams, and D is a set of edges (t_i, t_j) such that $t_i \cap t_j = \varnothing$. Figure 4 depicts an adjacency matrix of the graph generated for teams from Figure 3.

To allocate exactly experts to maximum number of teams, we find the maximum clique of graph G_D. Algorithm 3 we propose takes into account the graph features. Its inputs are matrix Δ and graph G_D, and its output is a maximum set *Allocate* of independent teams. It calculates an upper bound of the set size using (5) and calculates a lower bound by running the greedy Algorithm 1. Then, it orders the graph vertices by vertex power descending, and modifies G_D to G'_D.

Algorithm 3: Search for maximum clique in non-redundant team independency graph

Input: A matrix Δ of expert competences
Input: A graph $G_D = (T, D)$ of independency of non-redundant teams
Output: A subset $Allocate \subseteq T$ of independent teams representing problem solution

```
UpperBond ← Equation5 (Δ)
LowerBound ← GreedyAlgorithm (G_D)
T' ← Ordering (T)    G'_D ← Modifying (G_D, T')
Allocate ← LowerBound
if |LowerBound| = |UpperBond| then
    return Allocate
else
    for CliqueSize ← |LowerBound| + 1 to |UpperBond| do
        G''_D ← GenerateSubgraph (G'_D, CliqueSize)
        Clique ← SearchClique (G''_D, CliqueSize)
        if Clique = ∅ then
            return Allocate
        else
            Allocate ← Clique
return Allocate
```

Algorithm 4: Search for clique of required size in sub-graph

Input: A sub-graph $G''_D = (T'', D'')$
Input: A required size *CliqueSize* of clique
Output: A subset $Clique \subseteq T''$ of required size

```
for i ← 1 to |T''| do
  if |T''| – i < CliqueSize then
      return ∅
  else
    Stack(0).team ← t_i    Stack(0).count ← 1
    Stack(0).neighbours ← neighbourhood(t_i)    top ← 1
    while top > 0 do
        if top = CliqueSize then
            return GenerateClique(Stack)
        if Stack(top–1).count > |Stack(top–1).neighbours| then
            top ← top – 1
            if top > 0 then
                Stack(top–1).count ← Stack(top–1).count + 1
            continue
        nb ← Stack(top–1).neighbours(Stack(top–1).count)
        flag ← true
        for j ← 0 to top–1 do
            if nb ∉ Stack(j).neighbours then
                flag ← false    break
        if flag then
            Stack(top).count ← FirstNext(nb)
            Stack(top). neighbours ← neighbourhood(nb)
            Stack(top).team ← nb    top ← top + 1
        else
            Stack(top–1).count ← Stack(top–1).count + 1
```

The algorithm checks the equality of the lower and upper bounds and returns *Allocate* as optimum. Otherwise, it organizes a loop to find the largest team size from |*LowerBound*| + 1 to |*UpperBond*|. To speed up the search, function *GenerateSubgraph* reduces G'_D to G''_D of smaller size, *CliqueSize*, and function *SearchClique* finds a required clique.

Algorithm 4 searches for a clique of the required *CliqueSize* in sub-graph G''_D. It forms the clique by selecting a vertex from 1 to |T''|–*CliqueSize*+1 and adding other mutually adjacent vertices. To perform combinatorial search, it uses a *Stack*. All vertices pushed in the *Stack* are mutually connected. When the stack depth reaches *CliqueSize* the search is over and the clique is extracted from the stack. Otherwise, the algorithm checks if it has visited all neighbors of the vertex assigned to record *top* – 1. If yes, it pops the top record and returns to the previous vertex. If no, it passes to the next neighbor *nb*. If *nb* is adjacent to all previous vertices in the *Stack*, the algorithm pushes *nb* in the next record and repeats the described steps.

In Figure 4, the filled four rows and four columns describe the maximum clique that represents an optimal solution including four teams as follows: $t_{13} = \{p_0, p_7\}$, $t_{30} = \{p_1, p_6, p_9, p_{10}\}$, $t_{38} = \{p_2, p_4, p_8\}$ and $t_{40} = \{p_3, p_5\}$.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
t_0	1	1	1									
t_1	1	1		1								
t_2	1	1								1	1	
t_3	1	1										1
t_4	1		1			1						
t_5	1		1						1			
t_6	1			1			1					
t_7	1			1					1			
t_8	1				1		1			1		
t_9	1					1				1		
t_{10}	1					1						1
t_{11}	1						1			1	1	
t_{12}	1						1					1
t_{13}	1							1				
t_{14}	1								1	1		
t_{15}	1								1			1
t_{16}		1	1	1								
t_{17}		1	1		1							
t_{18}		1	1					1	1			
t_{19}		1	1						1		1	
t_{20}		1		1	1							
t_{21}		1		1			1					
t_{22}		1		1						1		
t_{23}		1			1			1				
t_{24}		1			1					1		
t_{25}		1			1							1
t_{26}		1				1		1	1			
t_{27}		1				1			1		1	1
t_{28}		1					1	1				
t_{29}		1					1		1	1		
t_{30}		1					1			1	1	
t_{31}		1					1					1
t_{32}		1						1	1	1		
t_{33}		1							1	1	1	
t_{34}			1	1				1				
t_{35}			1	1					1			
t_{36}			1		1	1						
t_{37}			1		1			1				
t_{38}			1		1				1			
t_{39}			1				1					
t_{40}				1		1						
t_{41}					1	1		1				
t_{42}					1	1				1		
t_{43}					1	1						1
t_{44}						1	1	1				
t_{45}						1	1			1		
t_{46}						1	1					1

Figure 3 – Non-redundant teams of experts from Table 2

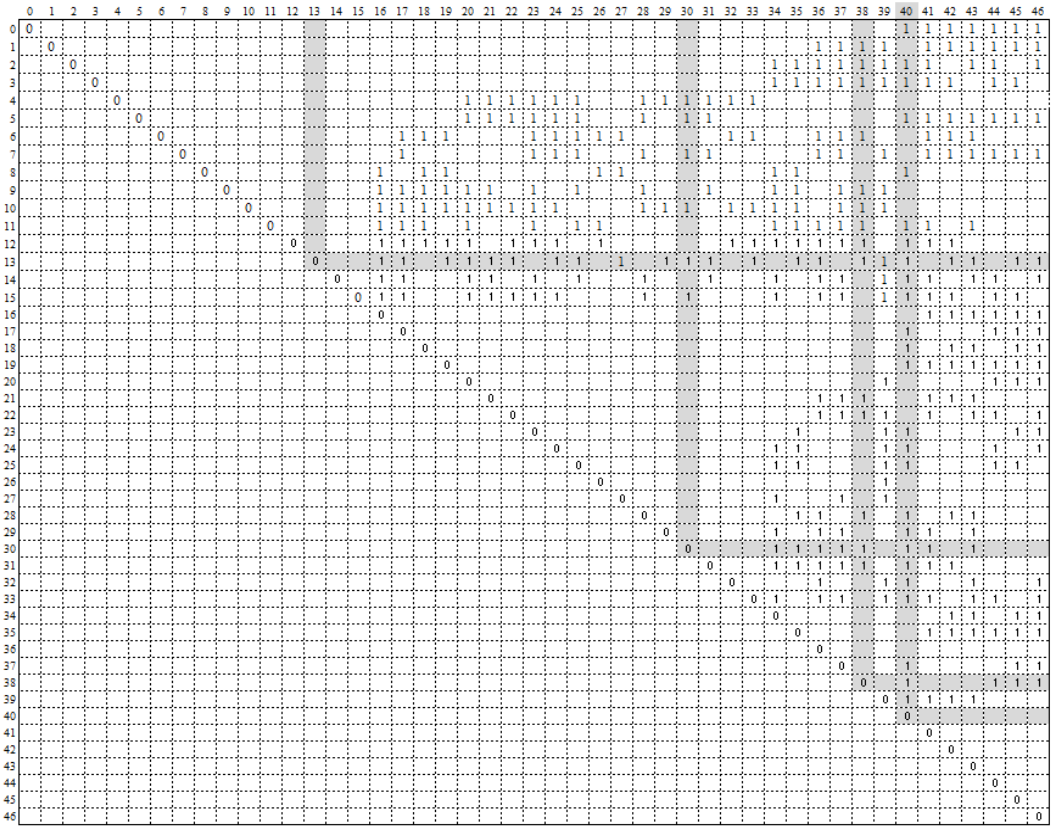

Figure 4 – Adjacency matrix of non-redundant teams independency graph G_D

Experimental results

We have developed a computer program that implements both the greedy and exact algorithms of allocating experts to teams. Table 4 reports experimental results obtained on six runs of the program on various expert samples of 20 programmers and 20 competences. The samples differ by minimum (upper bound of teams count) and average number of competences per expert (third and fourth parameters in the table). The increase of upper bound from 4 to 14 causes the growth of the maximum team count (exact solution) from 4 to 10, the greedy lower bound from 3 to 9, the competence count per expert from 5.8 to 15.8. The difference between the upper bound and the optimal solution has increased from 0 (run 1) to 4 (run 6).The greedy solution is one team less on average, although it is optimal for run 2. The number of generated non-redundant teams has increased from 665 to 930 and then has decreased to 204. The number of all teams (redundant and non-redundant) has been larger over the number of non-redundant teams by 70.8 downto 3.5 times.

Table 4. Experimental results of allocating 20 experts to programmer teams for 20 competences and various matrix Δ

Parameter	Run					
	1	2	3	4	5	6
Greedy lower bound	3	5	4	6	8	9
Maximum team count	4	5	6	7	9	10
Upper bound	4	6	8	10	12	14
Competences per expert	5.8	7.6	10.0	12.0	13.8	15.8
Non-redundant teams	665	930	857	513	377	204
All teams (times)	70.8	23.8	18.5	14.9	6.2	3.5

Conclusion

The paper has formulated a combinatorial problem of allocating experts to maximum set of programmer teams accounting for professional competences. In our work, to tackle the problem we have developed two algorithms: greedy heuristic and exact optimal. The first algorithm is fast and solves the problem using set cover problem solutions. Although the second algorithm is slow, it is a criterion for the evaluation of heuristic algorithm quality. The developed software allocates experts to teams and allows for obtaining experimental results on various-size input data. The fast double-greedy algorithm slightly loses to the exact algorithm by quality, but is applicable to large-size combinatorial problems.

REFERENCES

1. **Joshi, S.** Agile Development - Working with Agile in a Distributed Team Environment / **S. Joshi** // MSDN Magazine, 2012, Vol.27, No.1, pp.1-6.

2. **Collier, K.W., Agile Analytics:** A Value-Driven Approach to Business Intelligence and Data Warehousing. – Pearson Education, 2012. – 74 p.

3. **Muller, J.P., Rao, A.S., Singh, M.P.** A Teams: An Agent Architecture for Optimization and Decision-Support, Proceedings 5th International Workshop, ATAL'98 Paris, France, July 4–7, 1998, pp. 261-276.

4. **Masood Z., Hoda R., Blincoe K.** (2017) 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, vol 283. Springer, Cham. https://doi.org/10.1007/978-3-319-57633-6_19.

5. **R. Britto, P. S. Neto, R. Rabelo, W. Ayala and T. Soares,** "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.

6. Wrike [Электронный ресурс] – Режим доступа: https://www.wrike.com/, – Загл. с экрана – Яз. англ. Дата доступа – 28.10.2021.

7. Flow [Электронный ресурс] – Режим доступа: https://www.getflow.com/, – Загл. с экрана – Яз. англ. Дата доступа – 28.10.201

8. **Gutirrez, J. H., Astudillo, C. A., Ballesteros-P?rez, P., Mora-Meli?, D. and Candia-V?jar, A.** (2016) The multiple team formation problem using sociometry. Computers and Operations Research, 75. pp. 150-162. ISSN 0305-0548 doi: https://doi.org/10.1016/j.cor.2016.05.012

9. **Barricelli, N.A.** Symbio genetic evolution processes realized by artificial methods / **N.A. Barricelli** // Methodos, 1957, pp. 143–182.

10. **Prihozhy A.A., Zhdanouski A.M.** Method of qualification estimation and optimization of professional teams of programmers. «System analysis and applied information science». – 2018, No. 2, pages 4-11. https://doi.org/10.21122/2309-4923-2018-2-4-11

11. **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, 2019. – P. 305–310.

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

13. **Sijin, J.** Perspectives on Software, Technology and Business: Programmer Competency Matrix / **J. Sijin** // [Electronic resource]. –Mode of access: https://sijinjoseph.com/programmer-competency-matrix/. – Date of access: 28.10.2021.

14. **Karp R.M.** (1972) Reducibility among Combinatorial Problems. In: **Miller R.E., Thatcher J.W., Bohlinger J.D.** (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA. https://doi.org/10.1007/978-1-4684-2001-2_9

15. **Prihozhy, A.A.** Asynchronous scheduling and allocation / **A.A. Prihozhy** / Proceedings Design, Automation and Test in Europe. Paris, France. – IEEE, 1998, pp. 963-964.

16. **A. Prihozhy, S. Casale-Brunet, E. Bezati and M. Mattavelli.** "Efficient Dynamic Optimization Heuristics for Dataflow Pipelines," IEEE International Workshop on Signal Processing Systems, IEEE, pp. 337- 342, October 2018.

17. **Prihozhy, A., Casale-Brunet, S., Bezati, E., M. Mattavelli.** Pipeline Synthesis and Optimization from Branched Feedback Dataflow Programs. Journal of Signal Processing Systems, Springer Nature, 2020, Vol. 92, pages 1091–1099 https://doi.org/10.1007/s11265-020-01568-5.

ЛИТЕРАТУРА

1. **Joshi, S.** Agile Development - Working with Agile in a Distributed Team Environment / S. Joshi // MSDN Maga-zine, 2012, Vol.27, No.1, pp.1-6.

2. **Collier, K.W.,** Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. – Pearson Education, 2012. – 74 p.

3. **Muller, J.P., Rao, A.S.**, Singh, M.P. A Teams: An Agent Architecture for Optimization and Decision-Support, Pro-ceedings 5th International Workshop, ATAL'98 Paris, France, July 4–7, 1998, pp. 261-276.

4. **Masood Z., Hoda R., Blincoe K.** (2017) 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 Ex-treme Programming. XP 2017. Lecture Notes in Business Information Processing, vol 283. Springer, Cham. https://doi.org/10.1007/978-3-319-57633-6_19

5. **R. Britto, P. S. Neto, R. Rabelo, W. Ayala and T. Soares,** «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.

6. Wrike [Электронный ресурс] – Режим доступа: https://www.wrike.com/, – Загл. с экрана – Яз. англ. Дата доступа – 28.10.2021.

7. Flow [Электронный ресурс] – Режим доступа: https://www.getflow.com/, – Загл. с экрана – Яз. англ. Дата досту-па – 28.10.2021

8. **Gutirrez, J. H., Astudillo, C. A., Ballesteros-P?rez, P., Mora-Meli?, D. and Candia-V?jar,** A. (2016) The multiple team formation problem using sociometry. Computers and Operations Research, 75. pp. 150-162. ISSN 0305-0548 doi: https://doi.org/10.1016/j.cor.2016.05.012

9. **Barricelli, N.A.** Symbio genetic evolution processes realized by artificial methods / **N.A. Barricelli** // Methodos, 1957, pp. 143–182.

10. **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

11. **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, 2019. – P. 305–310.

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

13. **Sijin, J.** Perspectives on Software, Technology and Business: Programmer Competency Matrix / **J. Sijin** // [Electronic resource]. –Mode of access: https://sijinjoseph.com/programmer-competency-matrix/. – Date of access: 28.10.2021.

14. **Karp R.M**. (1972) Reducibility among Combinatorial Problems. In: **Miller R.E., Thatcher J.W., Bohlinger J.D.** (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA. https://doi.org/10.1007/978-1-4684-2001-2_9.

15. **Prihozhy, A.A.** Asynchronous scheduling and allocation / **A.A. Prihozhy** / Proceedings Design, Automation and Test in Europe. Paris, France. – IEEE, 1998, pp. 963-964.

16. **A. Prihozhy, S. Casale-Brunet, E. Bezati and M. Mattavelli.** "Efficient Dynamic Optimization Heuristics for Dataflow Pipelines," IEEE International Workshop on Signal Processing Systems, IEEE, pp. 337- 342, October 2018.

17. **Prihozhy, A., Casale-Brunet, S., Bezati, E., M. Mattavelli.** Pipeline Synthesis and Optimization from Branched Feedback Dataflow Programs. Journal of Signal Processing Systems, Springer Nature, 2020, Vol. 92, pages 1091–1099 https://doi.org/10.1007/s11265-020-01568-5.

Прихожий А.А.

ТОЧНЫЙ И ЖАДНЫЙ АЛГОРИТМЫ РАСПРЕДЕЛЕНИЯ ЭКСПЕРТОВ НА МАКСИМАЛЬНОМ МНОЖЕСТВЕ ГРУПП ПРОГРАММИСТОВ

Белорусский национальный технический университет

Распределение экспертов по программистским группам, отвечающее требованиям профессио-нальной компетенции в сфере программирования, специфицированным в ИТ-проекте, является сложной комбина-торной проблемой. В данной работе решается задача формирования максимального числа групп с включением в них экспертов, обеспечивающих выполнение каждой группой требований к компетенциям. В статье разрабаты-ваются и сравниваются два алгоритма решения задачи: эвристический жадный и точный оптимальный. Жадный алгоритм итеративно решает задачу о покрытии на матрице экспертных компетенций до тех пор, пока не смо-жет создать работоспособную группу из оставшихся экспертов. В статье доказано, что этот алгоритм не оп-тимален, даже если задача о покрытии решается оптимально. Алгоритм назначения экспертов является дважды жадным, если он использует жадный алгоритм покрытия множества. Предлагаемый точный алгоритм находит оптимальное решение на трех шагах: создание набора всех не избыточных групп, построение графа независимости групп и поиск максимальной клики графа. Алгоритм генерации групп обходит дерево поиска, построенное так, чтобы гарантировать нахождение всех не избыточных групп и поглощение всех избыточных групп. Ребра графа независимости групп соединяют вершины-группы, не имеющие общих экспертов. В статье предложен алгоритм поиска максимальной клики, учитывающий особенности графа и решаемой задачи. Экспериментальные результа-ты показывают, что точный алгоритм является оптимальным эталонным, а алгоритм двойной жадности явля-ется быстрым и может давать решение

Anatoly Prihozhy, received his Diploma of Electrical Engineering from the State Polytechnic, Minsk, Belarus in 1975, his PhD degree in computer-aided design from the National Academy of Sciences Minsk, Belarus in 1984, and his Doctor Habilitation degree in computer science from Ukraine, Kyiv and Belarus, Minsk in 1999. He was Visiting Professor at the Swiss Federal Institute of Technology, Lausanne, Switzerland in 2001, 2004, 2010 and 2013, 2015 and 2016 and at the Freiburg University, Germany in 2000. He is currently full professor at Computer and System Software Department of the Belarusian National Technical University. He has several books and more than 300 publications in Eastern and Western Europe, USA and Canada His research interests include programming, hardware and system description languages, compilers and tools, system-, high- and logic-level computer aided design and optimization of parallel and incompletely specified digital systems.