UDC 004.4-004.9

COMBINATORIAL PROBLEM OF ALLOCATING EXPERTS TO PROGRAMMER TEAMS

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

In the rapidly developing information technology industries, organizations and companies 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. In the literature, the process of allocating tasks in agile software development teams has not received much attention. In [4], the authors describe the process of task allocation as including three mechanisms of workflow across teams: team-independent, team-dependent, and hybrid workflow; and five types of task allocation strategies: manager-driven, team-driven, individual-driven, manager-assisted and team-assisted. In [5], the authors emphasize the relevance of the team in the agile methodologies: a successful agile 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 at every level, for various tasks and projects. In [9], a method for formalizing and evaluating the competency of individual programmers and entire programmer teams was proposed. The method evaluates the expertise of a programmer team taking into account the requirements for a particular project, including the constraints on average competency of programmers, the competency of best representatives on each technology; threshold competency of a programmer and a team. 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.

Table 1 – Sections of programmer competency matrix

Computer Science	**Software Engineering**
0. data structures	3. source code version control
1. algorithms	4. build automation
2. systems programming	5. automated testing
Programming	
6. problem decomposition	
7. systems decomposition	**Experience**
8. communication	21. languages with professional experience
9. code organization within a file	22. platforms with professional experience
10. code organization across files	23. years of professional experience
11. source tree organization	24. domain knowledge
12. code readability	
13. defensive coding	**Knowledge**
14. error handling	25. tool knowledge
15. IDE	26. languages exposed to
16. API	27. codebase knowledge
17. frameworks	28. knowledge of upcoming technologies
18. requirements	29. platform internals
19. scripting	30. books
20. database	31. blogs

This paper formulates a combinatorial problem of allocating a set of experts of programming languages, technologies and tools in the maximum number of programmer teams, assuming that each expert is assigned to one team. Expert is a programmer who has high level of competency and skills in at least one technology.

Let $C = \{c_1,\ldots, c_m\}$ be a set of 32 topics (listed in Table 1) Joseph Sijin proposed in [13] in order to create the programmer competency matrix and estimate the expertise of candidates to participating in IT projects. He formulated requirements to the programmer competency level on each of the topics and introduced a metric of four predefined levels $L0$, $L1$, $L2$ and $L3$. Let a certain IT project specifies requirements to competence over 12 topics described in Table 2: at least one member of each team that works on the project must have an expertise level larger than $L1$ for each of the competency topics. Such a member is considered as an expert regarding the corresponding competence within the project. The project requires that each team would include at least one expert on each competency. Programmers of required count who has lower competence level are allowed to be added to the project teams as well. However, a team is considered as unworkable if it has no expert on each competence topic.

Let $P = \{p_1,\ldots p_n\}$ be a set of programmers who have expressed his (her) desire to work on the project, have evaluated his (her) expertise level on each of the competency topics, C and filled in a questionnaire. As a result, a variable $Level(p, c)$, $p \in P$ and $c \in C$, describes the competency level of programmer p on topic c. Table 3 reports $Level(p, c)$ for 12 programmers and 12 competency topics. Observing the table rows and columns, we can conclude that the level of competency varies from $L0$ to $L3$. Considering individual experts as entities having advanced knowledge, experience and ability is crucial for the allocation of multi-skilled human resources to research and development projects. Observing Table 3 we conclude that each programmer has the expertise level of $L2$ and higher for at least one competency topic, therefore all 12 programmers are qualified as experts, which can constitute a core of working teams.

Let set C of competence topics be a universe. Let $S_p = \{c \mid c \in C$ and $Level(p, c) \geq L2\}$ for each $p \in P$ be a set of competences in which programmer p is an expert:. A collection $S = \{S_1,\ldots,S_n\}$ of sets of competences represents n experts. We describe the collection with a matrix $\Delta[n\times m]$. Element δ_{ij} of the matrix equals 1 if expert i has competence j at the required level, and equals 0 otherwise. Table 4 describes matrix Δ of the collection of competences for 12 experts at the constraint: $level \geq L2$. The right column of the table reports the number of competences each expert has. The bottom row reports for each competence the number of experts who obtain the competence.

Let Ω be a set of feasible allocations of experts to a set T of workable teams, assuming that the number of teams can vary in a wide range. Our objective is to solve the following problem:

$$\max_{T \in \Omega} |T| \tag{1}$$

subject to

$$\bigcup_{p \in T_i} S_p = C \quad for\ all\ T_i \in T \tag{2}$$

The following equation estimates an upper bound of the team count regarding the constraint on the competency level:

$$upper(|T|) = \min_{c \in C} \left[\sum_{p \in P} \delta_p \right] \tag{3}$$

If T^{max} is an accurate solution of problem (1), then $T^{max} \leq upper(|T|)$. According to Table 4 there are four experts who have the competence indexed by 0 of the $L3$ level, therefore equality $upper(|T|) = 4$ holds. It means the maximum number of teams T^{max} does not exceed four.

Given the universe, C and the collection, S of n sets, whose union equals the universe, the set cover problem is to identify the smallest sub-collection of S whose union equals the universe. Solving the problem gives a minimum subset $T_1 = Set\_Min\_Cover\ (C, S)$ of experts, which cover all competences of

Table 2 – Twelve competencies selected for setting up a project

Subsection	Level	Requirement
1. data structures	L0	Doesn't know the difference between Array and LinkedList
	L1	Able to explain and use Arrays, LinkedLists, Dictionaries etc in practical programming tasks
	L2	Knows space and time tradeoffs of the basic data structures, Arrays vs LinkedLists etc.
	L3	Knowledge of advanced data structures like B-trees, binomial and fibonacci heaps, tries etc.
2. algorithms	L0	Unable to find the average of numbers in an array
	L1	Basic sorting, searching and data structure traversal and retrieval algorithms
	L2	Tree, Graph, simple greedy and divide and conquer algorithms etc.
	L3	Able to code dynamic solutions, good knowledge of graph and numerical algorithms etc.

Table 2 continued

6. problem decomposition	L0	Only straight line code with copy paste for reuse
	L1	Able to break up problem into multiple functions
	L2	Able to come up with reusable functions/objects that solve the overall problem
	L3	Use of appropriate data structures and algorithms that encapsulate aspects of the problem
9. code organization within a file	L0	No evidence of organization within a file
	L1	Methods are grouped logically or by accessibility
	L2	Code is grouped into regions and well commented with references to other source files
	L3	File has license header, summary, well commented, consistent white space usage
11. source tree organization	L0	Everything in one folder
	L1	Basic separation of code into logical folders
	L2	No circular dependencies, binaries, libs, docs, builds all organized into folders
	L3	Physical layout of source tree matches logical hierarchy and organization
15. IDE	L0	Mostly uses IDE for text editing
	L1	Knows their way around the interface, able to effectively use the IDE using menus
	L2	Knows keyboard shortcuts for most used operations
	L3	Has written custom macros
16. API	L0	Needs to look up the documentation frequently
	L1	Has the most frequently used APIs in memory
	L2	Vast and In-depth knowledge of the API
	L3	Has written libraries that sit on top of the API to simplify frequently used tasks

Table 2 continued

21. languages with professional experience	L0	Imperative or Object Oriented
	L1	Imperative, Object-Oriented and declarative (SQL), weak vs strong typing etc.
	L2	Functional, added bonus if they understand lazy evaluation, currying, continuations
	L3	Concurrent (Erlang, Oz) and Logic (Prolog)
22. platforms with professional experience	L0	1
	L1	2-3
	L2	4-5
	L3	6+
23. years of professional experience	L0	1
	L1	2-5
	L2	6-9
	L3	10+
25. tool knowledge	L0	Limited to primary IDE (VS.Net, Eclipse etc.
	L1	Knows about some alternatives to popular and standard tools
	L2	Good knowledge of editors, debuggers, IDEs, open source alternatives etc. etc.
	L3	Has actually written tools and scripts, added bonus if they've been published
30. books	L0	Unleashed series, 21 days series, 24 hour series, dummies series…
	L1	Code Complete, Don't Make me Think, Mastering Regular Expressions
	L2	Design Patterns, Peopleware, Programming Pearls, Algorithm Design Manual etc.
	L3	Structure and Interpretation of Computer Programs, Concepts Techniques, Models of Computer Programming, Art of Computer Programming, Database systems etc.

Table 3 – Competence level of twelve programmers (case study)

Pro-gram-mer	Competence												Σ
	0	1	2	3	4	5	6	7	8	9	10	11	
0	L1	L3	L2	L0	L3	L0	L2	L0	L2	L0	L1	L0	L14
1	L3	L1	L2	L0	L1	L1	L3	L2	L3	L3	L0	L3	L22
2	L3	L2	L3	L3	L0	L1	L1	L0	L0	L2	L3	L3	L21
3	L3	L0	L3	L3	L3	L2	L2	L3	L0	L3	L0	L2	L24
4	L1	L2	L1	L0	L2	L3	L0	L2	L0	L3	L3	L2	L19
5	L1	L3	L1	L3	L3	L3	L0	L2	L1	L3	L0	L2	L22
6	L0	L2	L1	L0	L0	L2	L3	L0	L1	L2	L1	L0	L12
7	L3	L3	L3	L2	L0	L1	L3	L1	L3	L2	L1	L1	L23
8	L1	L3	L0	L0	L0	L0	L2	L1	L1	L3	L3	L0	L14
9	L1	L1	L0	L2	L2	L3	L3	L2	L3	L1	L1	L2	L21
10	L1	L1	L0	L2	L0	L1	L3	L0	L1	L3	L2	L2	L16
11	L	L2	L2	L0	L0	L2	L2	L1	L2	L1	L2	L2	L17
Σ	L19	L23	L18	L15	L14	L19	L24	L14	L17	L26	L17	L19	L225

Table 4 – Matrix Δ of collection of competences at constraint competence ≥ L2 (case study)

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

Level $L2$ and higher. Subset T_1 2282S represents a core of a team. The team may be extended by adding programmers of lower competence level. Removing from collection S the sets which correspond to experts of T_1 gives a reduced collection $S = S \setminus \{S_p\}$, $p \epsilon T_1$. We ask the question if a new workable team can be formed from experts that remain in S. To answer the question, we solve the set cover problem again and form a second team T_2 = *Set_Min_Cover* (C, S). If a covering solution exists, the T_2 team that is composed of experts who cover all competences is created, otherwise T_2 is empty and the process of forming the teams is over.

Algorithm 1 allocates experts to teams. The number of teams is initially unknown. The algorithm generates teams until the remaining set of experts is not able to meet the constraint on the competency level over all competences. If at least one competence is not covered, the experts cannot form a workable team. When the execution of Algorithm 1 is over, T represents the resulting set of created teams and R represents a set of experts, which have not been included in the workable teams. Initially $T = Ø$ and $R = S$. Boolean variable *Next_Team* controls the loop of generating the teams. Variable *Team* is a new team of smallest size generated by the procedure *Set_Min_Cover* (C, R). The procedure solves the set minimum cover problem and selects a minimum number of experts for the given constraint on competences. If the procedure has failed to generate a team, the set, *Team* is empty, and *Next_Team* is assigned false. Otherwise, the nonempty *Team* is added to set T, and collection R of competence sets is reduced by subtracting the sub-collection that corresponds to the experts of *Team*. When the loop execution is over, teams $T_1,\ldots,T_k$ are formed and the remaining collection R represents experts which cannot cover all competences of C. For this reason, the experts are included in a reserve team.

Table 5 – Stepwise allocation of experts to teams by Algorithm 1 (case study)

Team	Expert	Competences											
		0	1	2	3	4	5	6	7	8	9	10	11
Iteration 1													
T_1	3	1	0	1	1	1	1	1	1	0	1	0	1
	11	0	1	1	0	0	1	1	0	1	0	1	1
Iteration 2													
T_2	2	1	1	1	1	0	0	0	0	0	1	1	1
	9	0	0	0	1	1	1	1	1	1	0	0	1

Table 5 continued

Iteration 3													
T_3	4	0	1	0	0	1	1	0	1	0	1	1	1
	7	1	1	1	1	0	0	1	0	1	1	0	0
Iteration 4													
T_4	1	1	0	1	0	0	0	1	1	1	1	0	1
	5	0	1	0	1	1	1	0	1	0	1	0	1
	10	0	0	0	1	0	0	1	0	0	1	1	1
Iteration 5													
Re-serve	0	0	1	1	0	1	0	1	0	1	0	0	0
	6	0	1	0	0	0	1	1	0	0	1	0	0
	8	0	1	0	0	0	0	1	0	0	1	1	0

Table 5 describes the stepwise allocation of twelve experts to four workable teams the Algorithm 1 has generated in five loop iterations. The workable teams are as follows: $T_1 = \{3, 11\}$, $T_2 = \{2, 9\}$, $T_3 = \{4, 7\}$ and $T_4 = \{1, 5, 10\}$. It is easy to see that each workable team covers all twelve competences of set C. The remaining experts are included in team *Reserve* = {0, 6, 8}. This team does not cover competences 0, 3, 7, and 11. Therefore, it is unworkable.

In [14], Richard Karp proved that the set cover problem belongs to the NP-complete combinatorial problems. Therefore, Algorithm 1, which reduces the problem of allocating experts in teams to multiple solving the set cover problem, has the computational complexity that is at least the same as the set cover problem. It should be noted, that Algorithm 1 may find no exact solution in general case [15].

Conclusion

The paper has formulated a combinatorial problem of allocating experts to maximum number of programmer teams. It has evaluated the expert competences over the programmer competency matrix by taking into account project requirements. The proposed algorithm of solving the problem iteratively generates programmer teams with a minimum number of experts, thus trying to create the maximum number of teams. To minimize the number of experts in a team, the algorithm exploits the set cover problem, which is NP-complete. The example illustrates the formulated problem and proposed algorithm.

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. Müller, 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 – 28.10.2021

8. Gutiérrez, J. H., Astudillo, C. A., Ballesteros-Pérez, P., Mora-Melià, D. and Candia-Véjar, 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. Прихожий А. А., Ждановский А. М. Метод оценки квалификации и оптимизация состава профессиональных групп программистов. «Системный анализ и прикладная информатика». 2018; (2): 4–11. https://doi.org/10.21122/2309-4923-2018-2-4–11.

10. Прихожий, А. Эвристический генетический алгоритм оптимизации вычислительных конвейеров / А. А. Прихожий, А. М. Ждановский, О. Н. Карасик, М. Маттавелли // Доклады БГУИР, 2017, № 1, с. 34–41.

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: Research Paper Collection, Issue 3. – 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;(4):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 R. E. Miller; J. W. Thatcher (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103.

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.