# Scrum Task Allocation Based on Particle Swarm Optimization

Lucija Brezočnik*[0000−0002−3622−428X], Iztok Fister Jr.[0000−0002−6418−1272], and Vili Podgorelec[0000−0001−6955−7868]

Information Systems Laboratory, Institute of Informatics,
Faculty of Electrical Engineering and Computer Science, University of Maribor,
Koroška cesta 46, SI-2000 Maribor, Slovenia
`*Corresponding author: lucija.brezocnik@um.si`

**Abstract.** In this paper, we present a novel algorithm called STAPSO, which comprises Scrum task allocation and the Particle Swarm Optimization algorithm. The proposed algorithm aims to address one of the most significant problems in the agile software development, i.e., iteration planning. The actuality of the topic is not questionable, since nowadays, agile software development plays a vital role in most of the organizations around the world. Despite many agile software development methodologies, we include the proposed algorithm in Scrum Sprint planning, as it is the most widely used methodology. The proposed algorithm was also tested on a real-world dataset, and the experiment shows promising results.

**Keywords:** Agile Software Development, Particle Swarm Optimization, Scrum, Software Engineering, Task Allocation

## 1 Introduction

The idea of the iterative and agile development is all but new [1]. Ongoing changing priorities, desire to accelerate product delivery, the increase of productivity, improvement of project visibility, and enhancing software quality [2] are the top five reasons for adopting agile. Furthermore, in the report from Gartner Inc. [3], which is the world's leading research and advisory company, it is evident that the traditional project and development methods, e.g., waterfall, are evermore unsuitable [4, 5]. Consequently, we can state that agile software development is, nowadays, not a competitive advantage anymore, but rather the need for the organizations to survive on the market.

Regardless of the chosen agile method, e.g., Scrum, Kanban, Scrumban, XP (extreme programming), and Lean, monitoring of its performance must be carried out. Success in agile projects is most often measured by velocity in 67%, followed by the iteration burndown (51%), release burndown (38%), planned vs. actual stories per iteration (37%), and Burn-up chart (34%) [2]. However, a prerequisite for a successful monitoring of the progress is undoubtedly precise iteration planning. The latter is not only the number one employed agile technique in the organizations [2], but also one of the hardest tasks, as is evident from many scientific papers [6, 7] and interviews conducted with different CIOs (Chief Information Officers). Also, each task defined in a given iteration must be estimated precisely. The estimation can be conducted with various techniques [8, 2], e.g., number sizing (1, 2, . . . , 10), Fibonacci sequence (1, 2, 3, 5, 8, . . . ), and T-shirt sizes (XS, S, M, L, XL, XXL or XXXL). However, we must not forget about dependencies between tasks which result in the implementation order.

Thus, from an apparently simple problem arises a considerable optimization problem that is dealt with daily in organizations all around the world. When dealing with numerous dependencies and tasks, solving a problem by hand becomes very hard. On the contrary, we propose a systematical solution that is based on nature-inspired algorithms. Nature-inspired algorithms are a modern tool for solving hard continuous and discrete problems. They draw inspiration for solving such problems from nature. Until recently, more than 100 nature-inspired algorithms have been proposed in the literature [9], where Particle Swarm Optimization (PSO) [10] is one of the oldest and well-established nature-inspired algorithms. Many studies have proved theoretically and practically that PSO is a very simple, as well as efficient algorithm [11, 12] appropriate even for real-world applications [13].

In this paper, we show the modifications of the basic PSO algorithm that is applied to the problem of Scrum task allocation. The new algorithm, called STAPSO, is developed, implemented, and tested on a real dataset.

We believe that this is the first work that deals with the problem of Scrum task allocation in the optimization domain. Altogether, the purpose of this paper is to:

– represent Scrum task allocation as an optimization problem,
– propose the Particle Swarm Optimization algorithm for solving Scrum task allocation, or simply STAPSO, and
– test the proposed algorithm on a real dataset.

The structure of this paper is as follows: Section 2 outlines the fundamentals of Scrum, while Section 3 describes the fundamentals of the PSO algorithm, together with STAPSO algorithm. Section 4 presents the design of the experiment, along with the results in Section 5. The paper concludes with a summary of the performed work and future challenges.

## 2 Scrum

Scrum is the most used agile methodology, with 58% share of the market [2] and is by definition “a framework for developing, delivering, and sustaining complex products” [14, 15]. It consists of three primary roles, i.e. the Scrum Master, the Product Owner, and the Development Team. In the organizations, the Scrum Master is responsible for Scrum promotion and offers support regarding Scrum theory, values, and practices. Product Owner is a focal role since it is connected with the development team and the stakeholders. Two of his/her primary goals are to maximize the value of the product, and definition of the user stories from the product backlog. The remaining role, i.e., the Development Team, is liable for product increment delivery at the end of each Sprint. The Development Team is cross-functional and self-organizing, meaning that the people in it have all the skills required to deliver the product successfully.

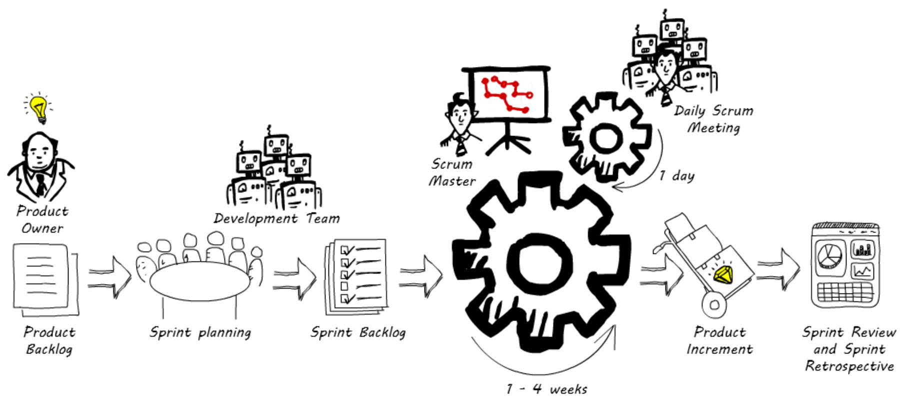

**Fig. 1.** The Scrum framework.

In Scrum, the process starts with the Product Owners' definition of the product backlog, which is a prioritized list of user stories (see Fig. 1). Afterwards, Sprint Planning starts. At this meeting, the team decides which user stories from the Product Backlog will be carried out in the upcoming Sprint (because the Product Backlog is prioritized, they pull user stories from the top of the list). The newly created document is called a Sprint Backlog and contains an in-depth description of the chosen user stories. After that, everything is ready for the beginning of the Sprint, that usually lasts between one and four weeks. Each day of the Sprint starts with a brief daily Scrum (short stand-up meeting) at which the Development Team exchanges opinions regarding the previous day and highlights possible problems. At the end of each Sprint, Sprint Review and Sprint Retrospective are carried out by the Development Team and the Product Owner, with the intention to find the potential for improvement.

For the calculation of the optimal line, it is necessary to determine the duration of the Sprint first ($n\_days$). For example, if we decide on the two week long Sprints, the Development Team actually has ten working days, assuming Saturday and Sunday are free days. After that, based on tasks from the Sprint Backlog, the total estimated effort ($t\_effort$) is obtained as their sum. Optimum per day ($opt\_d$) can now be calculated by Eq. 1.

$$opt\_d = \frac{t\_effort}{n\_days} \tag{1}$$

Ideal or optimal line ($Oline$) is derived from linear function $f(x) = ax + b$ and is presented by Eq. 2, where $x$ denotes the specific day of the Sprint.

$$Oline = -\frac{t\_effort}{n\_days} * x + t\_effort \tag{2}$$

Current line ($Cline$) is calculated by Eq. 3, where $Edone$ denotes the summarized effort of the tasks per given day.

$$Cline = Oline(x) - (Oline(x-1) - Edone) \tag{3}$$

## 3 Particle Swarm Optimization for Scrum task allocation

In this Section, the proposed algorithm called STAPSO is described in detail. Since the algorithm is based on the PSO, its explanation is divided into two Subsections. Subsection 3.1 depicts the fundamentals of the PSO, and Subsection 3.2 presents the proposed algorithm in detail.

### 3.1 Fundamentals of PSO

The PSO algorithm [10] preserves a population of solutions, where each solution is represented as a real-valued vector $\mathbf{x} = (x_{i,1}, \ldots, q_{i,D})^T$ for $i = 1, \ldots, Np$ and $j = 1, \ldots, D$, and the parameter $Np$ denotes the population size, and the parameter $D$ dimension of the problem. This algorithm explores the new solutions by moving the particles throughout the search space in the direction of the current best solution. In addition to the current population $\mathbf{x}_i^{(t)}$ for $i = 1, \ldots, Np$, also the local best solutions $\mathbf{p}_i^{(t)}$ for $i = 1, \ldots, Np$ are preserved, denoting the best $i$-th solution found. Finally, the best solution in the population $\mathbf{g}^{(t)}$ is determined in each generation. The new particle position is generated according to Eq. (4):

$$\begin{aligned}\mathbf{v}_i^{(t+1)} &= \mathbf{v}_i^{(t)} + C_1 U(0,1)(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}) + C_2 U(0,1)(\mathbf{g}^{(t)} - \mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)},\end{aligned} \tag{4}$$

where $U(0,1)$ denotes a random value in interval $[0,1]$, and $C_1$ and $C_2$ are learning factors. Algorithm 1 depicts the original PSO algorithm.

Interestingly, many surveys have recently revealed that the PSO algorithm was used in numerous real-world applications [13, 16]. However, the presence of the PSO algorithm in the software engineering research area is still in the minority.

In the next Subsection, the proposed STAPSO algorithm is presented for the Scrum task allocation problem.

**Algorithm 1** Pseudocode of the basic PSO algorithm

```
Input: PSO population of particles $\mathbf{x_i} = (x_{i1}, \ldots, x_{iD})^T$ for $i = 1 \ldots Np$, $MAX\_FEs$.
Output: The best solution $\mathbf{x}_{best}$ and its corresponding value $f_{min} = \min(f(\mathbf{x}))$.
 1: init_particles;
 2: $eval = 0$;
 3: while termination_condition_not_meet do
 4:    for $i = 1$ to $Np$ do
 5:       $f_i$ = evaluate_the_new_solution($\mathbf{x}_i$);
 6:       $eval = eval + 1$;
 7:       if $f_i \leq pBest_i$ then
 8:          $\mathbf{p}_i = \mathbf{x}_i$; $pBest_i = f_i$; // save the local best solution
 9:       end if
10:       if $f_i \leq f_{min}$ then
11:          $\mathbf{x}_{best} = \mathbf{x}_i$; $f_{min} = f_i$; // save the global best solution
12:       end if
13:       $\mathbf{x}_i$ = generate_new_solution($\mathbf{x}_i$);
14:    end for
15: end while
```

### 3.2 STAPSO algorithm

The following Section depicts the process of a Scrum task allocation problem using the STAPSO algorithm. For this problem, the following three modifications were applied to the basic PSO algorithm:

- representation of individuals,
- design of fitness function, and
- constraint handling.

**Representation of individuals** Candidate solutions in the basic PSO algorithm are represented as real-valued vectors $\mathbf{x}$, whilst a Scrum task allocation problem demands an integer vector $\mathbf{y}$ symbolizing the effort of a particular task. For that reason, mapping between representation of solutions in real-valued search space to the solution in a problem space is needed. In a STAPSO, this mapping is conducted in a similar manner as it was done for the problem of sport training sessions' planning [17]. A candidate solution in the proposed STAPSO algorithm is also represented, using the real-valued vector $\mathbf{x}_i = \{x_i0, \ldots, x_in\}^T$ for $i = 1 \ldots n$ with elements $x_{ij} \in [0, 1]$. In order to obtain effort values for fitness function calculation, firstly the permutation of task effort $\pi_i = \{\pi_{i1}, \ldots, \pi_{in}\}$ is mapped from the vector $\mathbf{x}_i$ such that the following relation is valid:

$$x_{i\pi_{i0}} < x_{i\pi_{i1}} < \ldots < x_{i\pi_{in}}. \quad (5)$$

Vector $y_i = \{y_{i0}, \ldots, y_{in}\}^T$ is determined from task description, Table 1. Table 2 presents an example of mapping the candidate solution $\mathbf{x}_i$ via permutation of task effort $\pi_i$ to the final task allocation.

**Table 1.** Task description table (example)

| **Task_ID** | **Effort** |
|---|---|
| 0 | 3 |
| 1 | 2 |
| 2 | 4 |
| 3 | 3 |
| 4 | 5 |

**Table 2.** Candidate solution mapping

| | **Dimension** $j$ | | | | |
|---|---|---|---|---|---|
| **Elements** $i$ | 0 | 1 | 2 | 3 | 4 |
| **Candidate solution** $\mathbf{x}_i$ | 0.70 | 0.42 | 0.21 | 0.94 | 0.52 |
| **Permutation** $\pi_i$ | 3 | 1 | 0 | 4 | 2 |
| **Task allocation** $\mathbf{y}_i$ | 3 | 2 | 3 | 5 | 4 |

**Fitness function** Fitness function is calculated according to Eq. 6 as follows:

$$f(x) = \left|\sum_{j=0}^{n\_days} (calculated\_effort\_per\_day_j)\right| \tag{6}$$

where *n_days* denotes the number of days, and *calculated_effort_per_day* is calculated effort for every day according to the constraint:

$$\begin{aligned}&\forall d \in \{1, 2, \ldots, n\_days\}, \forall t \in \{1, 2, \ldots, n\_tasks(d)\},\\ &\sum_{i=1}^{t} effort(i) \leq opt\_d\end{aligned} \tag{7}$$

where the variables $d$ and $t$ denote the current day of the Sprint, and the number of tasks per day, respectively. Final effort per day is then calculated as the sum of the tasks' efforts, that should not exceed the value of the *opt_d* (Eq. 1).

**Constraint handling** As discussed in previous Sections, there is sometimes a particular order (dependency) of some tasks. In other words, it means that one task must be completed before the beginning of another task. Most candidate solutions that are obtained according to mapping in Table 2 are unfeasible, i.e., they violate the dependency conditions. In our case, unfeasible solutions are penalized. Algorithm 2 presents our solution for handling constraints, where the function *is_violated*() checks if the dependency condition is violated. If the dependency condition is violated, the algorithm assigns a very high penalty [18] value to this particle. Despite many constraint handling methods, our penalization method behaves very well on the current problem. For that reason, we have not tested the behavior of any other constraint handling methods yet [19].

**Algorithm 2** Constraint handling in STAPSO

```
 1: $violations = 0$;
 2: $fitness = f(x)$;{calculated by Eq. 6}
 3: for $i = 1$ to $Num_{Rules}$ do
 4:     if $is\_violated()$ then
 5:         $violations = violations + 1$;
 6:     end if
 7: end for
 8: if $violations > 0$ then
 9:     $fitness = violations * 1000$;
10: end if
```

## 4 Experiment

The experimental approach was used in order to show the power of the STAPSO algorithm. Thus, Subsection 4.1 comprises parameter settings of the algorithm and the computer environment, and Subsection 4.2 presents test data, along with the constraints on Scrum tasks that were used in this study.

### 4.1 Setup

Experiments were conducted on an Intel XEON Z240 computer. STAPSO is implemented in the Python programming language without using any special software libraries. The algorithm ran on a Linux Ubuntu 16.04 operating system. After the extensive parameter tuning, the following parameter settings were used based on their best performance:

– population size $Np$: 75,
– dimension of the problem $D$: 60,
– number of function evaluations per run $MAX\_FEs = 30000$,
– total runs: 25,
– cognitive component $C_1 = 2.0$,
– social component $C_2 = 2.0$,
– velocity: [-4, 4],
– number of days: 10,
– number of Sprint: 1.

### 4.2 Test data and constraints

Table 4 presents test data that were used in this study. Test data for such experiments is very hard to get due to the company policy of confidential data. Thus, the source of test data is an internal project that was conducted within our laboratory. In Table 4, $Task\_ID$ denotes the identification number of a particular task, while *Effort* symbolizes the effort of this task. In this study, the following constraints were considered:

$$\Psi = \{(T7, T3), (T6, T22), (T4, T58), (T33, T31)\}.$$

Thereby, $\Psi$ denotes the implementation order of the tasks, i.e., task $T7$ must be implemented before task $T3$, task $T6$ must be implemented before task $T22$, etc. In the context of the algorithm, this means that all of the possible solutions must obey all of the given constraints and provide legitimate task allocation also considering Eq. 2 and Eq. 3.

## 5 Results

In the total of 25 runs, an optimal solution was found 20 times (i.e. success rate: 80%), meaning that no tasks were left behind for the next Sprint. In three cases (12%), the algorithm did not allocate one task with the estimated effort of 1, and in two cases (8%), the algorithm did not allocate one task estimated with the effort of 2. We want to highlight that all constraints presented in Subsection 4.2 were satisfied in all runs. On average, an optimal solution was found after 5533 function evaluations.

Table 3 comprises an in-depth description of one optimal solution. The latter presents the sequence of tasks' implementation for one Sprint, which is described with the Task IDs (column 2) and belonging tasks' effort (column 3). Per each day, the number of tasks and remaining effort is recorded, respectively.

**Table 3.** Example of task allocation from Table 4 (optimal solution)

| Day | Tasks allocated | Tasks' effort | Number of tasks | Effort remaining |
|---|---|---|---|---|
| 1 | 4, 12, 15, 17, 21, 32, 42 | 5, 3, 1, 1, 1, 2, 2 | 7 | 0 |
| 2 | 43, 27, 49, 48, 58, 33 | 3, 3, 3, 1, 1, 4 | 6 | 0 |
| 3 | 51, 7, 50, 5 | 1, 5, 4, 5 | 4 | 0 |
| 4 | 24, 26, 45, 35, 57, 54, 25 | 2, 1, 2, 2, 4, 2, 2 | 7 | 0 |
| 5 | 18, 10, 29, 16 | 2, 5, 3, 5 | 4 | 0 |
| 6 | 6, 22, 8, 53, 31 | 5, 4, 3, 1, 2 | 5 | 0 |
| 7 | 28, 44, 19, 0, 30, 3 | 4, 2, 1, 2, 4, 2 | 6 | 0 |
| 8 | 1, 14, 20, 37, 40, 52, 23, 38 | 2, 2, 2, 1, 1, 3, 3, 1 | 8 | 0 |
| 9 | 56, 34, 41, 11, 2, 9, 13 | 2, 3, 1, 2, 1, 3, 3 | 7 | 0 |
| 10 | 36, 39, 46, 47, 55, 59 | 3, 2, 4, 2, 2, 2 | 6 | 0 |
| $\sum$ | **60** | **150** | **60** | **0** |

Fig. 2 and Fig. 3 present the same proposed solution of the STAPSO algorithm, where two tasks with the estimated effort of 1 were not allocated. A non-optimal solution was chosen deliberately for easier explanation of the results and deviations. Fig. 2 presents the solution in the form of the burndown chart, and Fig. 3 shows allocated tasks per day of the Sprint.

In Scrum, a burndown chart is one of the most frequently used graphs to present the current state of the work of the project [2, 20]. On the abscissa axis (Fig. 2), days of the Sprint are displayed, and on the ordinate axis, the remaining

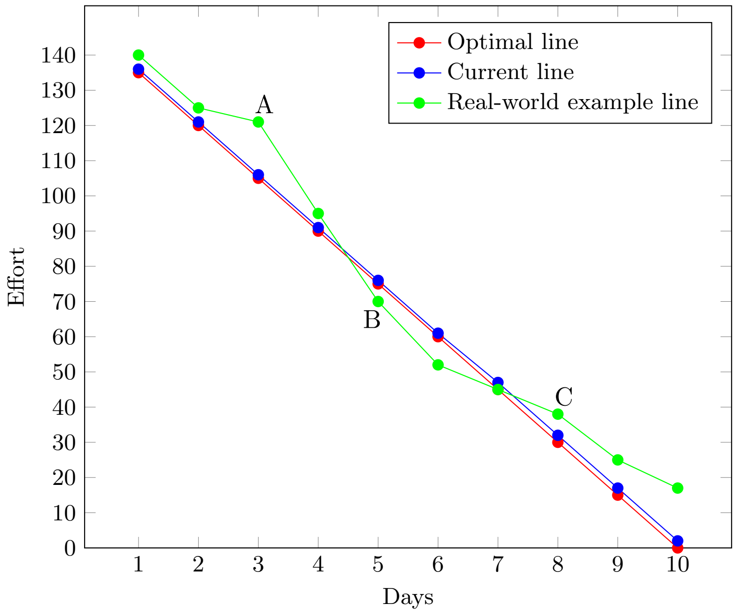

**Fig. 2.** Burndown chart of non-optimal solution (chosen deliberately for easier explanation of the results)

effort of the Sprint. The preparation of such a graph is carried out in several stages. Firstly, the optimal line is drawn. The optimal line is calculated with Eq. 2 and shows an ideal or optimal course of the implementation of the tasks. In Fig. 2, this line is colored with red. As stated before, all tasks are estimated with effort (see Table 4) and with their fulfillment, we can monitor remaining effort in a Sprint. If we look at the first day of the Sprint in Fig. 2, we can see that ideal effort completed per day is 15 (calculated with Eq. 1). Thus, the Development Team should, on their first day, complete tasks with the sum of the effort of at least 15. As we can see from Fig. 3, algorithm STAPSO for the first day allocated 5 tasks with the estimated effort sum of 14, meaning that, after the first day, the Development Team is one effort behind the optimal line (see blue line). In a real-world scenario, we can witness lines that are similar to the green line. From the latter, it is evident that the Development Team was behind the optimal line for the first four days, and on day 3 (point A) they fulfilled tasks with the effort sum of only 4. However, after the third day, the fulfillment of tasks went very quickly, so in two days they caught up and were in front of the optimal line on day five (point B). Point C shows that the progress has stopped on day 8 (they were behind the optimal line again), and they stayed behind it until the end of the Sprint.

In Fig. 3 the days of the Sprint show the allocated tasks given by the STAPSO algorithm. As we have said in the description of Fig. 2, the optimal effort sum

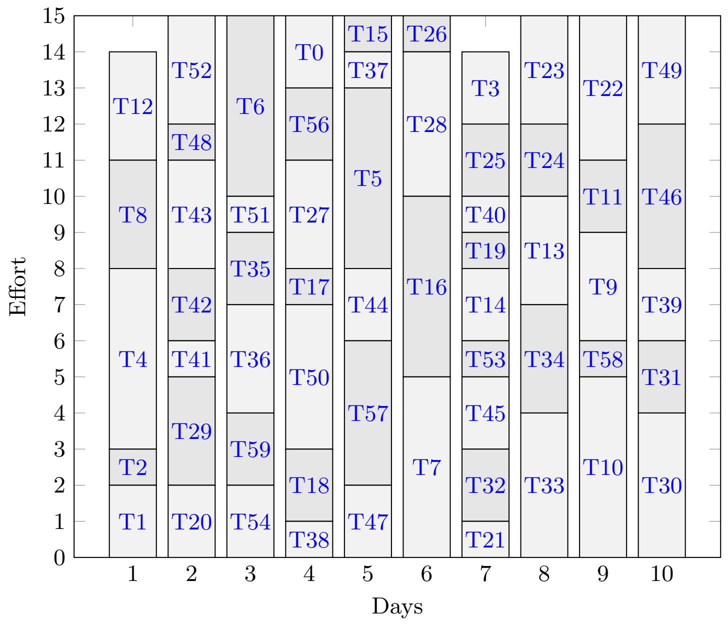

**Fig. 3.** Task allocation of non-optimal solution (chosen deliberately for easier explanation of the results)

per day is 15 (maximum value of the ordinate axis). This sum is also the value that the algorithm is trying to achieve per day. If we look at the results, on the first day the STAPSO algorithm allocated five tasks, i.e., $T1$, $T2$, $T4$, $T8$, and $T12$ (see Table 4), with the sum of effort of 14. On the second day, a sum of effort of 15 is fulfilled with the tasks $T20$, $T29$, $T41$, $T42$, $T43$, $T48$, and $T52$, etc. This kind of graph is beneficial for the Development Team and the Product Owner, since they have allocated tasks from the beginning of the Sprint.

## 6 Conclusion

A novel algorithm STAPSO was implemented and tested successfully on a real dataset. It offers a solution to the global problem of task allocation in the agile software development. The STAPSO algorithm can be applied to all of the known estimation techniques, e.g. number sizing, Fibonacci sequence, and T-shirt planning. Furthermore, it can be included in companies regardless of their size and maturity degree.

In the design of the algorithm, there is still significant room for improvement. In the future, we intend to study the impact of various constraint handling methods and variants of PSO on the behavior of the STAPSO algorithm. Hybridization of STAPSO with any other well-established algorithms, e.g., Differential Evolution is also a sparkling way for future research.

Since we have not found any similar algorithms for Scrum task allocation that are based on nature-inspired algorithms yet, we believe that this study could be a stepping stone for more links between the vibrant agile software development research area and optimization.

## Acknowledgment

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

## References

1. Takeuchi, H., Nonaka, I.: The New New Product Development Game. Harvard Business Review **64** (1986) 137–146
2. VersionOne: VersionOne 11th Annual State of Agile Report. (2017)
3. Kyte, A., Norton, D., Wilson, N.: Ten Things the CIO Needs to Know About Agile Development. Technical report, Gartner, Inc. (2014)
4. Gandomani, T.J., Nafchi, M.Z.: Agile transition and adoption human-related challenges and issues: A grounded theory approach. Computers in Human Behavior **62** (2016) 257–266
5. Chen, R.R., Ravichandar, R., Proctor, D.: Managing the transition to the new agile business and product development model: Lessons from cisco systems. Business Horizons **59**(6) (2016) 635–644
6. Heikkil, V.T., Paasivaara, M., Rautiainen, K., Lassenius, C., Toivola, T., Jrvinen, J.: Operational release planning in large-scale scrum with multiple stakeholders a longitudinal case study at f-secure corporation. Information and Software Technology **57** (2015) 116–140
7. Barney, S., Ke Aurum, A., Wohlin, C.: A product management challenge: Creating software product value through requirements selection. Journal of Systems Architecture **54** (2008)
8. Usman, M., Mendes, E., Weidt, F., Britto, R.: Effort estimation in agile software development: A systematic literature review. In: Proceedings of the 10th International Conference on Predictive Models in Software Engineering. PROMISE '14, NY, USA, ACM (2014) 82–91
9. Fister Jr., I., Yang, X.S., Fister, I., Brest, J., Fister, D.: A brief review of nature-inspired algorithms for optimization. Elektrotehniški vestnik **80**(3) (2013) 116–122
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. Volume 4., IEEE (1995) 1942–1948
11. Shi, Y., et al.: Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 Congress on evolutionary computation. Volume 1., IEEE (2001) 81–86
12. Yang, X.S.: Nature-inspired metaheuristic algorithms. Luniver press (2010)
13. Zhang, Y., Wang, S., Ji, G.: A comprehensive survey on particle swarm optimization algorithm and its applications. Mathematical Problems in Engineering (2015)
14. Sutherland, J.V., Sutherland, J.J.: Scrum: the art of doing twice the work in half the time. 1st edn. Currency (2014)

15. Schwaber, K., Sutherland, J.: The Scrum Guide. (2017)
16. Pluhacek, M., Senkerik, R., Viktorin, A., Kadavy, T., Zelinka, I.: A review of real-world applications of particle swarm optimization algorithm. In: International Conference on Advanced Engineering Theory and Applications, Springer (2017) 115–122
17. Fister, I., Rauter, S., Yang, X.S., Ljubič, K., Fister Jr., I.: Planning the sports training sessions with the bat algorithm. Neurocomputing **149** (2015) 993–1002
18. Coello, C.A.C.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer methods in applied mechanics and engineering **191**(11) (2002) 1245–1287
19. Mezura-Montes, E., Coello, C.A.C.: Constraint-handling in nature-inspired numerical optimization: past, present and future. Swarm and Evolutionary Computation **1**(4) (2011) 173–194
20. Cooper, R.G., Sommer, A.F.: Agile-stage-gate: New idea-to-launch method for manufactured new products is faster, more responsive. Industrial Marketing Management **59** (2016) 167–180

## Test data

**Table 4.** Test data

| Task_ID | Effort | Task_ID | Effort |
|---|---|---|---|
| T0 | 2 | T30 | 4 |
| T1 | 2 | T31 | 2 |
| T2 | 1 | T32 | 2 |
| T3 | 2 | T33 | 4 |
| T4 | 5 | T34 | 3 |
| T5 | 5 | T35 | 2 |
| T6 | 5 | T36 | 3 |
| T7 | 5 | T37 | 1 |
| T8 | 3 | T38 | 1 |
| T9 | 3 | T39 | 2 |
| T10 | 5 | T40 | 1 |
| T11 | 2 | T41 | 1 |
| T12 | 3 | T42 | 2 |
| T13 | 3 | T43 | 3 |
| T14 | 2 | T44 | 2 |
| T15 | 1 | T45 | 2 |
| T16 | 5 | T46 | 4 |
| T17 | 1 | T47 | 2 |
| T18 | 2 | T48 | 1 |
| T19 | 1 | T49 | 3 |
| T20 | 2 | T50 | 4 |
| T21 | 1 | T51 | 1 |
| T22 | 4 | T52 | 3 |
| T23 | 3 | T53 | 1 |
| T24 | 2 | T54 | 2 |
| T25 | 2 | T55 | 2 |
| T26 | 1 | T56 | 2 |
| T27 | 3 | T57 | 4 |
| T28 | 4 | T58 | 1 |
| T29 | 3 | T59 | 2 |