order to store it in the entire distributed database to make it consistent.

b) Competitors download the problem and (eventually) the algorithm provided by the proponent and try to resolve it. They can set up their own algorithms, or use existing ones, regardless of the type (i.e., exact methods, heuristics, metaheuristics etc.), store them into the database and share the obtained solutions on the marketplace.

c) Competitors post their solutions: (i) the one who posts the best solution for the problem up to that moment get rewarded; (ii) those who post solutions as open-source – regardless whether they are the best ones – get rewarded for having shared them with everyone. The reward is bounded by the gamified rewarding rules in section 2.3; (iii) those who want to post solutions with the intent of selling it – and are not the best ones for the problem up to that moment – must pay an amount of tokens to do it.

d) When nodes perform actions on the network or on the database, e.g., they post, sell or buy a solution or transfer their ownership, those are validated by other peers to guarantee the integrity and the ownership of each operation.

Figure 2 represents the ideal workflow previously outlined using a simple UML sequence diagram. The diagram outlines the scenario in which a proponent posts a problem and a competitor resolves it and posts the solution as open-source (or the solution result is the best one for the problem). If the solution is not the best one for the problem and it is not open-source, arrow at step 7 must be reversed and labelled "pay to public solution as private".

## 3. Example: Competing for the Knapsack Problem

The Knapsack Problem (KP) is a well-known NP-hard problem: given a set of items, each with a weight and a value, determine which item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports. In this section we show an example of the application of Genetic Algorithm (GA) to the Knapsack Problem [6] and how competitors can take advantage from the properties of techniques such as GA and compete to find the "best so far" solution for such a problem.

Genetic Algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). GAs are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals which are less fit will be eliminated. This means that the genesfrom the highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment. A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimisation terms, each individual in the population is encoded into a string or chromosome which represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are given opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., children), which share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (generational approach) or replace less fit individuals (steady-state approach). The basic steps of a simple GA are: (1) generate the initial solution; (2) valuate fitness of individuals in the population; (3) select parents from population; (4) recombine parents to produce children; (5) evaluate fitness of the children; (6) replace some or all of the population by the children. The evaluation-selection-reproduction cycle (steps 3 to 6) is repeated until a satisfactory solution is found or a stop criterion is reached.

### 3.1. Problem representation and objective function

The one-dimensional Knapsack Problem can be formulated as follows:

$$maximise \sum_{i=1}^{n} c_i x_i \tag{1}$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

wherein $n$ is the number of items; $c_i$ and $w_i$ are the value and the weight of the $i$-th item, respectively; $W$ is the maximum capacity; and $x$ is a integer variable that indicates the possibility that an item is included

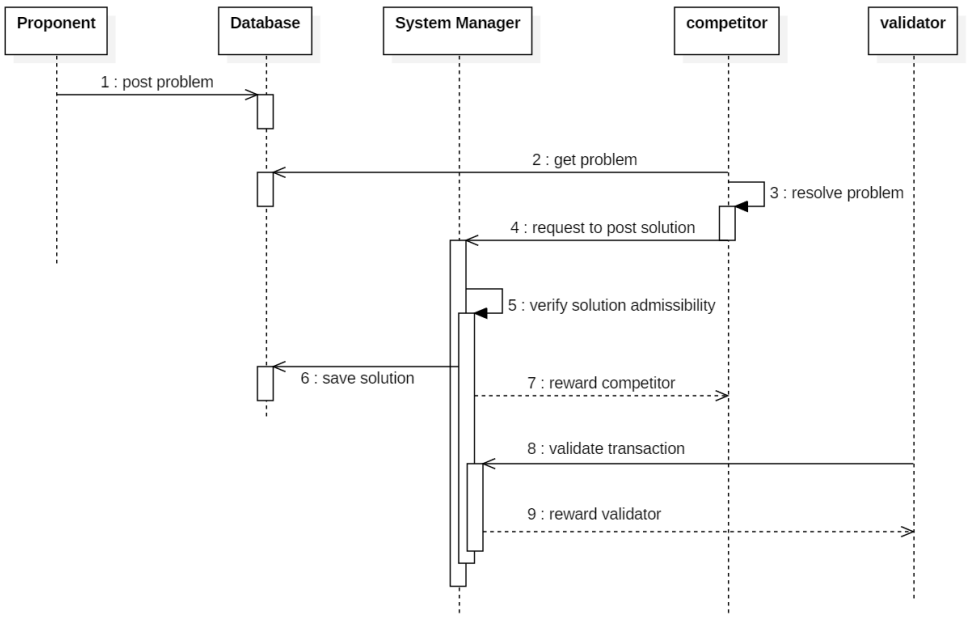


**Figure 2.** Gamified blockchain-based marketplaces: a workflow – notation loosely based on UML sequence diagrams.

| *i* | 1 | 2 | 3 | 4 | 5 | … | *n*-1 | *n* |
|---|---|---|---|---|---|---|---|---|
| *S[i]* | 0 | 1 | 0 | 0 | 1 | … | 0 | 1 |

**Figure 3.** Binary representation of a KP solution.

in the collection. In this example we consider the 0-1 Knapsack problem, therefore $x_i \in \{0, 1\}$ is a binary value, meaning that each item can be included zero or one times in the collection.

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme, i.e., a way to represent individuals in the GA population. The standard GA 0-1 binary representation is an obvious choice for the one-dimensional Knapsack Problem since it represents the underlying 0-1 integer variables. Hence, in this representation, we used a $n$-bit binary string, where $n$ is the number of variables in the KP. A value of 0 or 1 at the $i$-th bit implies that $x_i = 0$ or 1 in the solution, respectively. This binary representation of an individual's chromosome (solution) for the KP is illustrated in Figure 3. Note that a bit string $S \in \{0, 1\}^n$ might represent an infeasible solution. An infeasible solution is one for which the constraint (2) is violated, i.e., $\sum_{i=1}^{n} w_i x_i > W$ .

## 3.2. Initial Population

To achieve sufficient diversification one can randomly generate an initial population with a high size being fixed (e.g. $n = 100$) and construct each of the initial feasible solutions by a primitive primal heuristic that repeatedly randomly selects a variable and sets it to one if the solution is feasible. Another competitor can generate the initial population changing its size and using a simple constructive greedy or a clever heuristic to construct initial solutions rather than randomly generate them. Competitors can initialize their population as they want as long as the final posted solution is feasible for the problem.

## 3.3. Parent Selection

Parent selection is the task of assigning reproductive opportunities to each individual in the population. Typically in a GA we need to generate two parents who will have (one or more) children. The tournament selection method works by forming two pools of individuals, each consisting of $t$ individuals drawn from the population randomly. Using a larger value for $t$ has the effect of increasing selection pressure on the more fit individuals. A competitor can adopt the standard binary tournament selection method (i.e., $t = 2$) as the method for parent selection because it can be implemented very efficiently. Another competitor can base its selection on other criteria such as Roulette Wheel Selection where the probability for an individual to be selected is proportioned to its fintess; Rank Selection; Random Selection, etc.

## 3.4. Crossover and Mutation

The binary, problem-independent, representation adopted for the KP in this example allows a wide range of the standard GA crossover and mutation operators

to be adopted. Competitors can choose among several crossover operators. In a *one-point crossover*, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs. *Multi-point crossover* is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs. In *uniform crossover* the chromosome is not divided into segments, rather it treats each gene separately. Two parents have a single child. Each bit in the child solution is created by copying the corresponding bit from one or the other parent, chosen according to a binary random number generator [0, 1]. If the random number is a 0, the bit is copied from the first parent, if it is a 1, the bit is copied from the second parent. There exist a lot of other crossovers like Partially Mapped Crossover (PMX), Order based crossover (OX2), Shuffle Crossover, Ring Crossover, etc. Once a child solution has been generated through crossover, a mutation procedure is performed that mutates some randomly selected bits in the child solution, i.e., causes these chosen bits to change from 0 to 1 or vice versa. The rate of mutation is generally set to be a small value (in the order of 1 or 2 bits per string). However, each competitor can define the mutation value she desires.

### 3.5. Repair operator

Clearly, the child solution generated by the crossover and mutation procedures may not be feasible because the Knapsack constraints may not all be satisfied. In order to guarantee feasibility, competitors can apply several heuristics, such as a simple greedy algorithm.

### 3.6. Stopping Criterion

The following three kinds of termination conditions have been traditionally employed for GAs [16]: (i) an upper limit on the number of generations is reached; (ii) an upper limit on the number of evaluations of the fitness function is reached; or (iii) the chance of achieving significant changes in the next generations is excessively low. However, there are a lot of other criteria defined in the literature, all with their pros and cons. Competitors can choose among them or define their own stopping criteria. A simple stopping criterion could be based simply on the execution time a competitor is willing to spent.

### 3.7. Algorithmic outline

Settings of the GA heuristic a competitor can use for the KP are:

- the binary tournament selection method;
- the uniform crossover operator;
- a mutation rate equal to 2 bits per child string;
- to discard any duplicate children (i.e., discard any child which is the same as a member of the population);
- the steady-state replacement method based on eliminating the individual with the lowest fitness value.

This settings can be included as solution information when a competitor posts the solution generated by this algorithm set up. Of course, competitors can use simpler heuristics than GAs and also obtain better results, depending on the problem instance and search technique used.

Let's consider the data in Table 1 and a weight limit of 20. In this case there exist two optimal solutions that can be found with different algorithms. The optimal solutions consist of selecting the 2nd and 7th item or the 7th and 8th item, and are generated following the binary representation showed before: (0,1,0,0,0,0,1,0) and (0,0,0,0,0,0,1,1) both with a weight of 19 and a total value of 8. The solutions are then evaluated by the validation algorithm provided by the proponent, with the purpose to check their feasibility and execute the objective function to measure their quality. Once accepted the system provides to insert them into the database, together with the algorithm information, and register them on the blockchain whose transactions are validated by other peers to guarantee the integrity and the ownership of those operations as mentioned in section 2.5. At this point, people on the network can buy, sell or exchange those solutions and the algorithm set up through smart contracts that establish and guarantee the conditions to be respected for their use, payment and their eventual ownership transfer. For example, competitors could have generated intermediate solutions such as (0,1,1,0,0,0,0,0) or (0,0,0,0,0,1,1,0) of value 7 and weight 18 and 20, respectively. Those solutions could be acquired, through smart contracts, by other peers to solve similar problems or, for example, to use them as part of the initial population in a evolutionary algorithm. Finally, instead of competing, competitors could also cooperate starting from a initial population and share computational efforts to create new generations (i.e., children) in a distributed fashion and split the rewards. Smart contracts guarantee that each competitor will be rewarded by the effort spent to find the final solution and all the information will be stored on the blockchain.

## 4. Discussion

The problems we want people to solve on our marketplace often apply in areas where there is no

| item | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| value | 5 | 3 | 4 | 6 | 3 | 1 | 5 | 3 |
| weight | 15 | 8 | 10 | 14 | 11 | 9 | 11 | 8 |

**Table 1.** Example of data for the KP.

existing best solution. This is particularly true in the engineering. It is natural in the resolution of new problems of engineering fields that there are, at first, moments characterized by enthusiasm, early important results and the excitement that goes with them. However, this excitement is often struck down by the fact that no optimum solution is known, and the actual results could be not the best ones. Clearly, this is the case of search-based optimization problems, that involve search techniques to find the best solution in a huge space of possible solutions. There are three key ingredients for the application of search-based optimization to a widely number of applications: (i) the choice of the representation of the problem; (ii) the definition of the objective function; and (iii) a set of manipulation operators. Typically, a proponent will have a suitable representation for her problem, so the first of the pre-requisites is easily satisfied. Even the objective function is defined by the proponent, which is posted together with the problem representation and its description. However, the objective function can be proposed by competitors and eventually accepted by the proponent, for example through a stacking mechanism. Competitors, for their parts, can define manipulation operators or using existing ones. Different search techniques use different operators. As a minimum requirement, it will be necessary to mutate an individual representation of a candidate solution to produce a representation of a different candidate solution. It will make it possible to apply hill climbing approaches and certain forms of evolutionary computation. If it is possible to determine the set of "near neighbors" of a candidate solution (in term of its representation) then simulated annealing and tabu search can be applied. If, instead (or in addition), it is possible to sensibly cross-over two individuals (to produce a "child" which retain characteristics of both "parents") then genetic algorithms will be applicable. With these three ingredients it becomes possible to implement search algorithms. The results of the search algorithms can be compared, for example using random search or other algorithms provided by the competitors or by the proponent itself, to provide as baseline data. Naturally, the aim is to beat them, though in some areas even a purely simple, unsophisticated algorithm, such as random search, has been found to be not without value, even beating human-directed search in some cases. This fact is supported by the "no free lunch" (NFL) theorems that establish that for any algorithms any elevated performance over one class of problems is exactly paid for in performance over another class [20]. Of course, the goal is to find the best solution for a given problem, regardless from the used algorithm. However, having different algorithms to compare – and solutions generated by these algorithms – allows researchers and people to understand and balance the results with performance, with the possibility to create an ontology of problems and related algorithms together with their best solutions and information.

There exist a lot of search-based optimization algorithms and techniques. Although competitors can use precise optimization algorithms such as linear programming[6] to solve problems – and we encourage them to do so, whenever possible –, those are straightforward deterministic algorithms. Even though modern solvers can deal with thousand of variable and millions of clauses, these deterministic optimization algorithms are often inapplicable because the problems have objectives that cannot be characterized by a set of linear equations. Often there are multiple criteria and complex objective functions. Many of the optimization problems are augmented versions of known NP-complete problems and, as such, they are well suited to the application of metaheuristic search techniques. A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines of strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework [18]. A metaheuristic is not an algorithm. Rather, it is a consistent set of ideas, concepts, and *operators* that can be used to design heuristic optimization algorithms. This acknowledgment brings with it all of issues that are associated with the application of metaheuristic search techniques: (i) global optimum; (ii) predictability; and (iii) computational expense.

**Global Optimum.** There is no guarantee that the global optimum will be found. In many applications, there is a threshold, above which a solution will be "good enough" for purpose. Furthermore, optimization may not seek to find an optimal solution to a problem, but rather, it may seek to improve upon the current situation.

**Predictability.** Each execution will potentially yield different results. It is true that each execution of a metaheuristic search algorithm can yield different results, but all search algorithms are formulated in such a way that repeated executions can only improve on a

[6] Linear programming is a mathematical optimization technique that is guaranteed to locate the global optimum solution subject to some linear expression in the decision variables given as input to the linear programming model.

"best so far" result, rather than overturning a previous result. The algorithms may be terminated at any time and also after any number of executions to yield results which are the "best so far".

**Computational Expense.** Many individual candidate solutions may need to be considered before an acceptable quality solution is found. The kinds of problems to which search techniques seem to be readily applicable, are those where the solution is highly complex. While speedy answers may be attractive, they are not essential in many applications. To overcome computational expense, competitors can solve problems in a distributed fashion, sharing their computational power and split the rewards, as nowadays happen with the famous "mining pools". Several techniques can be used that involve parallelism (e.g., Multiple Threads, Island Models, Master-Slave Fitness Assessment) [12] and different reward approaches exist (e.g., slush's pool, pay-per-share, p2pools, etc.).

An attractive field to which search-based optimization techniques have been widely applied is software engineering [8, 9]. Many of the problems faced by software engineers turn out to have natural counterparts as "standard" optimization problems. Indeed, from its formal definition to this date a huge number of software engineering problems have been mathematically formulated as optimization problems and tackled with a considerable variety of search-based techniques. Often, of course, there are some modifications and enhancements that are required and suitable representations and objective functions must be formulated for each problem; therein lies interesting and exciting research.

## 5. Conclusions and Future Work

The role of blockchain technology to structure effective, trust-based and cooperative software engineering solutions is not clear yet but offers ample potential and great opportunity. The solution we outlined in the previous pages combines state of the art blockchain technologies in a new way and includes in the midst all the elements of gamification in a purposeful way such that a business-savvy software-based solution can be structured. Although a proof-of-concept of our proposal is currently under way of prototyping, we are looking to formalize our proposal using more structured approaches to software and requirements engineering, properly exploring the solution space defined by our proposed design pattern and understanding its factual feasibility beyond the theoretical illustration and discussion we currently offered. In the future we plan to carry out such formalization, addressing in particular the business benefits behind the solution possibly by means of industrial empirical software engineering research. From a technical perspective, we aim at eliciting the different technical implementation options currently existing to support blockchains (e.g., Ethereum, Hyperledger) and comparatively analyze their fitness for purpose towards designing for, implementing, and operating prototypes for the proposed idea. We plan to propose a software-centric business model behind the proposed solution such that the proposed design pattern may be accompanied by a sound and well-thought business plan template around which companies and practitioners interested in blockchain technology may base their own solutions.

Finally, although this framework promotes sharing open-source solutions, others rules can apply. We also aim at creating a platform that allows proponents to create contests and define their own rules, in accordance with the design pattern presented, defining a distributed ledger model that marks the progress of problem solving as intellectual capital.

## Acknowledgements

The authors are grateful to Hervé Gallaire for many useful feedbacks and insights for the previous versions of this paper.

## References

[1] Bell, R.M. and Koren, Y. (2007) Lessons from the netflix prize challenge. *SIGKDD Explorations 9* **2**: 75–79.

[2] Bengtsson, M. and Kock, S. (2000) âĂİ coopetitionâĂİ in business networksâĂŤto cooperate and compete simultaneously. *Industrial marketing management* **29**(5): 411–426.

[3] Burke, E.K., Curtois, T., Kendall, G., Hyde, M., Ochoa, G. and Vazquez-Rodriguez, J.A. (2009) Towards the decathlon challenge of search heuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (ACM): 2205–2208.

[4] Buterin, V. *et al.* (2014) A next-generation smart contract and decentralized application platform. *white paper* .

[5] Caddy, T. (2005) Tamper resistance. *In Encyclopedia of Cryptography and Security, edited by Henk C. A. van Tilborg: Springer* .

[6] Chu, P.C. and Beasley, J.E. (1998) A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics* **4**(1): 63–86.

[7] Dalpiaz, Fabiano; Ali, R. and Brinkkemper, S. (2018) Special section on gamification and software engineering. *Information & Software Technology* **95**: 177–178.

[8] Harman, M. and Jones, B.F. (2001) Search-based software engineering. *Information and software Technology* **43**(14): 833–839.

[9] Harman, M., Mansouri, S.A. and Zhang, Y. (2012) Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* **45**(1): 11.

[10] Katz, D. (2014) Transitive credit as a means to address social and technological concerns stemming

from citation and attribution of digital products. *Journal of Open Research Software* **2.1**.

[11] LaToza, T.D. and van der Hoek, A. (2016) Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE Software 33* **1**: 74–80.

[12] Luke, S. (2013) *Essentials of Metaheuristics* (Lulu), 2nd ed., 99–107. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[13] Morschheuser, B., Hamari, J. and Koivisto, J. (2016) Gamification in crowdsourcing: A review .

[14] Nakamoto, S. (2008) Bitcoin: A peer-to-peer electronic cash system .

[15] Olague, G. (2006) Gecco-2006 highlights. human competitive awards the humies. synthesis of interest point detectors through genetic programming. *SIGEVOlution 1* **3**: 28–29.

[16] Safe, M., Carballido, J., Ponzoni, I. and Brignole, N. (2004) On stopping criteria for genetic algorithms. In *Brazilian Symposium on Artificial Intelligence* (Springer): 405–413.

[17] Scalabrino, S., Geremia, S., Pareschi, R., Bogetti, M. and Oliveto, R. (2018) Freelancing in the economy 4.0: How information technology can (really) help. *Social Media for Knowledge Management Application in Modern Organizations* : 290–314.

[18] Sörensen, K. and Glover, F.W. (2013) Metaheuristics. In *Encyclopedia of operations research and management science* (Springer), 960–970.

[19] Tamburri, D.A., Lago, P. and van Vliet, H. (2013) Uncovering latent social communities in software development. *IEEE Software 30* **1**: 29–36.

[20] Wolpert, D.H. and Macready, W.G. (1997) No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* **1**(1): 67–82.

[21] Zheng, Z., Xie, S., Dai, H., Chen, X. and Wang, H. (2017) An overview of blockchain technology: Architecture, consensus, and future trends. *Big Data (BigData Congress), 2017 IEEE International Congress on* : 557–564.