Multi-hard Problems in Uncertain Environment

Michał Roman Przybyłek
mrp@pjwstk.edu.pl
Polish-Japanese Academy of Information Technology
Warsaw, Poland

Adam Wierzbicki
adamw@pjwstk.edu.pl
Polish-Japanese Academy of Information Technology
Warsaw, Poland

Zbigniew Michalewicz
zbyszek@cs.adelaide.edu.au
School of Computer Science
The University of Adelaide
Adelaide, Australia

ABSTRACT

Real-world problems are usually composed of two or more (potentially NP-Hard) problems that are interdependent on each other. Such problems have been recently identified as "multi-hard problems" and various strategies for solving them have been proposed. One of the most successful of the strategies is based on a decomposition approach, where each of the components of a multi-hard problem is solved separately (by state-of-the-art solver) and then a negotiation protocol between the sub-solutions is applied to mediate a global solution. Multi-hardness is, however, not the only crucial aspect of real-world problems. Many real-world problems operate in a dynamically-changing, uncertain environment. Special approaches such as risk analysis and minimization may be applied in cases when we know the possible variants of constraints and criteria, as well as their probabilities. On the other hand, adaptive algorithms may be used in the case of uncertainty about criteria variants or probabilities. While such approaches are not new, their application to multi-hard problems has not yet been studied systematically. In this paper we extend the benchmark problem for multi-hardness with the aspect of uncertainty. We adapt the decomposition-based approach to this new setting, and compare it against another promising heuristic (Monte-Carlo Tree Search) on a large publicly available dataset. Our comparisons show that the decomposition-based approach outperforms the other heuristic in most cases.

Keywords

Traveling Thief Problem; Co-evolution; Heuristics; Metaheuristics; Multi-objective optimization; Non-separable problems; Real-world optimization problems

1. INTRODUCTION

Real-world optimization problems have been studied and solved in the past, but the work resulted in solutions tailored to individual problems that could not be generalized. The reason for this limitation was the lack of appropriate models for the systematic study of salient aspects of real-world problems. It was argued in [16] that there are five main reasons behind the hardness of real-world problems: the descriptional size of the problem, modeling issues, noise, constraints, and some psychological pressures on the designer, when problems are big. In [15] real-world problems were categorized into two groups: (1) design/static problems, and (2) operational/dynamic problems. The first category includes a variety of problems, such as Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), Knapsack Problem (KP), etc. The second category includes problems presented in the now-a-days industries (e.g. real-world supply chain). The author claimed that, although some of the design/static problems (first category) are really hard and most of the current research has been concentrated on those problems, the problems from the second category (i.e. operational/dynamic problems) are much harder and represent a huge opportunity for Evolutionary Algorithms. It was also stated that the value of addressing the problems in the first category does not have significant influence on solving the problems in the second category [15].

One aspect of operational/dynamic problems: i.e. multi-hardness was studied in [5], [20], where by a multi-hard problem we mean a problem that is a non-trivial combination of classical (single) hard problems. The authors argued that most of real-world problems are really multi-hard (where several potentially NP-Hard problems interact with each other), while most of the current researches have been concentrated on single-hard problems (Traveling Salesman Problem, Knapsack Problem, Job Shop Scheduling, Vehicle Routing Problem, etc). Also, it was stated that *interdependency* among components in multi-hard problems plays a key role in the complexity of the problems. However, this interdependency is not found in single-hard problems [4].

The above observations led to the formulation of an abstract double-hard problem, called the Travelling Thief Problem (TTP) in [4], which is a non-trivial composition of two well-studied classical problems: the Travelling Salesman Problem and the Knapsack Problem. The authors have obtained some insights into the difficulty of multi-hard problems in general through an evaluation of algorithms for solving TTP. The goal was to compare known heuristics against algorithms that aim to exploit the structure of a multi-hard problem.

A lot of great work has been done in developing algorithms for single-hard problems, and the best algorithms are tuned to specific problems. Unfortunately, such problem-

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4206-3/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908812.2908814

specific algorithms are extremely sensitive to changes in the formulation of the problem, therefore are of no use when some modifications are introduced (e.g. new constraints are added). However, instead of throwing out our knowledge, and instead of building new algorithms from scratch, it may be possible to use existing algorithms as building blocks for solving multi-hard problems. In [5] the authors have developed the idea of CoSolver and applied it to TTP obtaining some promising results. The main idea behind CoSolver is to decompose a multi-hard problem into sub-problems, solve the sub-problems separately with some communication between the sub-problems, and then compose the solutions back to obtain a solution to the initial problem (Figure 1). In [20] CoSolver has been compared against a meta-heuristic that we have thought of as most promising for multi-hard problems: a Monte-Carlo Tree Search algorithm. Both algorithms have been also compared to exact solutions for a variety of instances of TTP, differing in difficulty and structure. Further, CoSolver has been extended by incorporating heuristics instead of exact solvers for the TSP and KP components of TTP. This extension has greatly improved the scalability of CoSolver without compromising quality.

The aim of this paper is to address one additional aspect of real-world problems: uncertain environment. We extend the benchmark problem for multi-hardness (the Traveling Thief Problem) with a new aspect of uncertainty. Then we adapt the decomposition-based approach to this new setting, and compare it against another promising heuristic (Monte-Carlo Tree Search). To make the comparison sound, we prepare a large publicly available dataset of generic instances for the Traveling Thief Problem and its extensions.

The structure of the paper is as follows. In the next section, we discuss related work. Section 3 formally defines the Probabilistic Traveling Thief Problem. In Section 4 we briefly recall Ordered Weighting Aggregation operators, which are used in Section 7 to compare various behaviours of our algorithms. Section 5 describes algorithms for multi-hard problems in uncertain environments: CoSolver algorithm, and Monte-Carlo Tree Search. Section 6 describes the benchmark instances for our model problem. Section 7 presents results of experiments with solving benchmarks using proposed algorithms. Section 8 concludes the paper.

2. RELATED WORK

Traveling Thief Problem (TTP) was first introduced in [4] as an example of multi-component optimization problem. It was presented as a combination of two well-known problems (Travelling Salesman Problem and Knapsack Problem). Authors showed that finding optimum solutions for each sub-problem do not guarantee finding the global solution, as the interdependency between the two problems influences the optimum solutions for the whole problem.

New algorithm (called CoSolver) for dealing with multi-hard problems was introduced in [5], where communications and negotiation processes between partial solutions were studied. Authors proposed also a simple heuristic (called Density-based Heuristic, DH) as a second approach. These two methods were compared on some generated benchmark TTPs. Results showed that the performance of CoSolver was much better than DH. In [20] various variants of CoSolver have been compared against classical approach based on Monte Carlo Tree Search.

In [4] it was argued that most of real-world problems are multi-hard problems, while most of the current research has been concentrated on single-hard problems (Traveling Salesman Problem, Knapsack Problem, Job Shop Scheduling Problem (see: [6], [9], and [26]), Vehicle Routing Problem (see: [25], etc.). Also, it was stated that interdependency among components in multi-hard problems plays a key role in the complexity of these problems. However, this interdependency is not found in designed single-hard problems.

Early attempts in solving multi-hard problems (described as large scale optimization problems) include Newton's methods and conjugate gradient methods [10], the partitioned quasi-Newton method [11], and linear programming [3]. However, a major drawback of these methods is their dependency on the algebraic formulation of a problem and the availability of gradient information. For many real-world problems where an algebraic formulation is intractable, simulation is often used instead to obtain the evaluation of a potential solution, by providing an output value for a given set of input decision variable values. This sort of black-box optimization is commonly seen in engineering and many other disciplines. For black-box optimization problems, meta-heuristics such as evolutionary algorithms (EAs) have a significant advantage over the traditional derivative-dependent optimization methods. Meta-heuristics do not rely on gradient information, and are less likely to be stuck on local optima because of their use of a population of candidate solutions. Furthermore, recent development in the field of meta-heuristics show that cooperative co-evolutionary EAs hold great promise as it was shown in [29], [13], and [14] for such problems. Nevertheless, significant challenges remain. Finally, there are also modern research on bi-level (and, general, multi-level) optimization, where optimization problems consist of several sub-components related by some hierarchical dependencies (see: [7] and [23]). In such setting, however, hierarchically lower components do not take into account the optimization tasks of upper components.

Uncertain environment may affect the objective function and constraints. As a result optimal solution vary over time. Different approaches have been proposed to deal with dynamic optimization problems. Evolutionary algorithms and swarm intelligence methods have been shown to be particularly useful for handling such problems due to their self-organizing nature [22]. Dealing with dynamically changing constrains, on the other hand, is still a challenge of major significance [21].

To solve such complex issues human computational potential can be used [12]. This is completely new approach to solve such problems and at the same time it is promising and interesting direction of the research. Teams of human decision makers and new heuristic algorithms could improve solutions of these problems.

3. THE MODEL: A MULTI-HARD PROBLEM WITH UNCERTAINTY

Let us recall first Traveling Thief Problem (TTP). It consists of n cities (i.e. nodes of a graph) together with a function d assigning to every pair i, j of cities a non-negative distance $d(i, j)$ from i to j ($d(i, j)$ plays the role of a weighted edge from a node i to a node j in the graph; $d(i, j) = +\infty$ means that there is no edge from i to j), and m items (each has a profit p_i and a weight w_i) distributed in the cities.

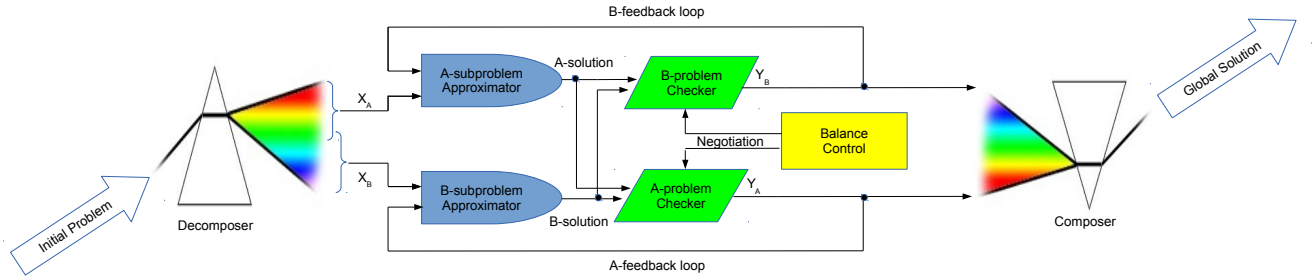

Figure 1: Decomposition of a problem on two sub-problems: A and B.

The availability of items is given by a function:

$$a(i) = \{c_1^i, \ldots, c_{k_i}^i\}$$

which shows that item i is available at cities:

$$c_1^i, c_2^i, \ldots, c_{k_i}^i$$

where c_j^i belongs to the set of cities $\{1, \ldots, n\}$. It follows from our notation, that k_i copies of item i can be found over all cities. Additionally, we are given the following constants:

- a non-negative real number R — the rent ratio;
- a natural number W — the capacity of the knapsack;
- two positive real numbers $v_{min} \leq v_{max}$ — the minimal and maximal speed of "the traveler".

The task is to find a complete tour Π visiting each city exactly once (i.e. a permutation Π of the initial segment of n positive natural numbers; abusing notation, we shall also write Π_{n+1} for Π_1; furthermore we assume that the permutation is stable on the first city — i.e. $\Pi_1 = 1$) and pick items from the cities x_i in a way that the total weight of the picked items does not exceed W, and the total profit (formulated in Eq. (1)) is maximized.

$$P = \sum_{i=1}^{m} p_i[x_i \neq 0] - R \sum_{i=1}^{n} t_{i,i+1} \tag{1}$$

where P is the total profit, $x_i \in a(i) \cup \{0\}$ represents the city that the item i should be picked from (0 refers to not picking the item at all), $[x_i \neq 0]$ is the Iverson bracket (i.e. equals 1 if $x_i \neq 0$, and 0 otherwise) and $t_{i,j}$ is the time to travel from Π_i to Π_j assuming the weight of all picked items by city Π_i is W_{Π_i}, which is given by the formula:

$$t_{i,j} = \frac{d(\Pi_i, \Pi_j)}{v_{max} - W_{\Pi_i} \frac{v_{max} - v_{min}}{W}} \tag{2}$$

where W_{Π_i} is the total weight of the picked items from cities $\{\Pi_2, \Pi_3, \ldots, \Pi_i\}$ (we assume that items from city $\Pi_1 = 1$ are picked at the end of the tour). Note that, if more items are picked while their total weight is smaller than W, the value of $t_{i,j}$ grows which causes reducing the value of P in Eq. (1). Also, by taking better tours in terms of total distance, some possibly high quality items (items which have a high profit) might only be available at the beginning of the tour and, hence, by picking those items, the travel time increases (items should be carried for a longer time), which causes reduction in the value of P. This shows that the interdependency between the two problems influences the optimum solutions for the whole problem.

In Probabilistic Traveling Thief Problem (TTP-PROB) each item is available to the thief at a certain probability. The availability of items is given by a function:

$$a(i) = \{[c_1^i, q_1^i], \ldots, [c_{k_i}^i, q_{k_i}^i]\}$$

where c_j^i is the number of a city, and q_j^i is the probability that item i is available in city c_j^i — i.e. when the thief enters a city c_j^i, the i-th item is available with probability q_j^i. Just like in TTP, the task is to maximize the total profit given in Eq. (1). This time, however, this profit depends not only on the strategy of the thief, but also on a particular scenario — which items are available when the thief enters a city, and which are not. Therefore, there may be various goals, a thief wants to achieve. For example, one goal could be to maximize the expected total profit. Another important goal could be the minimisation of the risk of the total profit. Such goals will be characterised by so-called "ordered weighted averaging aggregation operators" (OWA), which we describe in Section 4.

4. AGGREGATION OPERATORS

Ordered Weighted Averaging (OWA) operators are commonly used for decision-making under partial or complete ignorance or for equitable optimization [28], [27].

The motive behind selecting the OWA operator for aggregation of outcomes under uncertainty is the capability of OWA to encompass a range of operators from minimum to maximum, including various averaging operators like arithmetic mean. The OWA operator provides a flexibility to incorporate decision maker's preferences towards risk, including risk propensity (risk seeking) and risk aversion (avoiding risk) [24], [17]. The OWA operation involves three steps: (1) reordering of the input parameters; (2) determining the weights associated with the OWA operators; and (3) aggregation process.

The first step of the OWA aggregation is ordering the outcomes (given by a vector $[r_1, r_2, \ldots, r_n]$) from the worst to the best. For example, for maximized outcomes (as in our case), the outcomes should be ordered from smallest to largest. Let π be permutation of $\{1, 2, \ldots, n\}$ such that:

$$r_{\pi(1)} \leq r_{\pi(2)} \leq \cdots \leq r_{\pi(n)}$$

The OWA operator is a function $F\colon \mathbb{R}^n \to \mathbb{R}$ that can be expressed as:

$$F([r_1, r_2, \ldots, r_n]) = \sum_{i=1}^{n} w_i r_{\pi(i)} \tag{3}$$

where for every i we have $0 \leq w_i \leq 1$, the sum $\sum_{i=1}^{n} w_i$

equals one, and π is like described above. Therefore an OWA operator is completely determined by a vector of weights $w = [w_1, w_2, \ldots, w_n]$. Classical examples include:

- the function choosing the maximal element is an OWA operator corresponding to weights $[0, 0, \ldots, 1]$,
- the function choosing the minimal element is an OWA operator corresponding to weights $[1, \ldots, 0, 0]$,
- the function taking the average of elements is an OWA operator corresponding to weights $[\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}]$
- the function taking the median of elements is an OWA operator corresponding to weights $w_{\frac{n}{2}} = 1$ and $w_k = 0$ where $k \neq \frac{n}{2}$ for even n; and $w_{\frac{n-1}{2}} = \frac{1}{2}, w_{\frac{n+1}{2}} = \frac{1}{2}$ and $w_k = 0$ where $k \notin \{\frac{n-1}{2}, \frac{n+1}{2}\}$ for odd n.

Another interesting OWA operator takes its weights from normalised binomial coefficients:

$$\frac{1}{2^{n-1}}\left[\binom{n-1}{0}, \binom{n-1}{1}, \ldots, \binom{n-1}{n-1}\right]$$

This is a "smoother" version of the median operator — it prefers values that are close to the median. For this reason we call this OWA operator "typical". Two more useful OWA operators correspond to weights that:

- $\frac{1}{H_{n,p}}\left[\frac{1}{1^p}, \frac{1}{2^p}, \ldots, \frac{1}{n^p}\right]$ (p-th risk-avoiding operator)
- $\frac{1}{H_{n,p}}\left[\frac{1}{n^p}, \frac{1}{(n-1)^p}, \ldots, \frac{1}{1^p}\right]$ (p-th risk-seeking operator)

where $H_{n,p} = \sum_{k=1}^{n} \frac{1}{k^p}$ is the generalised harmonic number of order p.

Algorithm 1: Monte-Carlo Tree Search for TTP-PROB

```
Π ← ∅, K ← ∅
C^Π ← prepare c random initial scenarios
for i ← 1 to n do
    M ← possible extensions of partial cycle Π
    for m ∈ M do
        P_m ← −∞
        for k ← 1 to maxIter do
            Π' ← extend Π followed by m to a random cycle
            K' ← extend K with random items
            according to scenario C^Π[k mod c]
            P' ← profit(Π', K')
            if P > P_m then
                Π_m ← Π'
                P_m ← P
    Π ← extend partial cycle Π with such m* ∈ M
    that maximizes estimated profit P_m*
    C^Π ← prepare c random scenarios compatible with Π
    I^m* ← set of items available at city m*
    for t ∈ I^m* do
        S ← 0
        for k ← 1 to maxIter do
            Π' ← extend Π to a random cycle
            K' ← extend K with item t and random items
            according to scenario C^Π[k mod c]
            K'' ← extend K with random items according
            to scenario C^Π[k mod c]
            P' ← profit(Π', K')
            P'' ← profit(Π', K'')
            if P' > P'' then S ← S + 1
            else S ← S − 1
        if S > 0 then K ← K ∪ {t}
return Π, K
```

5. ALGORITHMS

In this section we show how to extend two most promising algorithms for TTP described in [20] to algorithms for TTP-PROB. One is a meta-heuristic, which is based on Monte-Carlo Tree Search. The other bases on the idea of CoSolver developed in [5] and [20].

5.1 Monte-Carlo Tree Search

The idea behind this heuristic is based on Monte-Carlo Tree Search [1]. Starting from the initial city and the empty knapsack we interchangeably perform the following two steps:

- (TSP Phase) extend the current route Π by a node m and run a number of random simulations with the extended route; the simulations are performed on c random scenarios C^Π that are compatible with the knowledge of item availability at the current route Π; calculate the best profit p_m from all simulations; add to the route node m^* having maximal profit p_{m^*}
- (KP Phase) for every set item t that is available at the current city, extend the knapsack by t and run a number of random simulations with the extended knapsack; the simulations are performed on c random scenarios C^Π that are compatible with the knowledge of item availability at the current route ; add to the knapsack items t that increases the expected total profit P.

until a complete tour is constructed (Alg. 1). The set C^Π consists of scenarios randomly generated according to the probabilities from a given instance, plus two additional scenarios: the optimistic scenario (all items with non-zero probabilities are available) and the pessimistic scenario (no items are available expect these with probability one). In the implementation, MCTS uses a relatively small number $8 \leq c \leq 16$ of randomly generated "trail" scenarios. One reason for this choice was to keep the running time of MCTS comparable to the running time of CoSolver, but more importantly we found that adding more random scenarios did not lead to any significant improvement of the solutions. Another variant of the algorithm could use a fixed set of scenarios C^Π and share it in all iterations, but we observed that our approach leads to a slightly more stable results due to a better diversity in the scenarios.

5.2 Decomposition Algorithms

In [5] the authors identified two sub-problems of TTP — one corresponding to a generalization of TSP, called Traveling Salesman with Knapsack Problem (TSKP), and another corresponding to a generalization of KP, called Knapsack on the Route Problem (KRP). Roughly speaking, TSKP corresponds to the part of TTP where we know which items are picked at which cities and the task is to find the optimal route, whereas KRP corresponds to the part of TTP where we are given a route of the thief, and the task is to find

the optimal picking of the items. Here is a more detailed description.

Traveling Salesman with Knapsack Problem (TSKP) consists of n cities, a distance function d, a positive integer W (the capacity of the knapsack), a non-negative real number R (the rent rate) and a function w assigning to every node $i \in \{1, 2, \ldots, n\}$ the total weight w_i of items picked at i, and two positive real numbers $v_{min} \leq v_{max}$ corresponding to the minimal and the maximal speed of the traveler. The task is to find a complete tour Π visiting each city exactly once, such that the following is minimized:

$$T = -R \sum_{i=1}^{n} t_{i,i+1} \tag{4}$$

where $t_{i,i+1}$ is defined in the same way as in Eq. (2) from Section 3.

Knapsack on the Route Problem (KRP) consists of n cities, a function d assigning to a city i the cost $d(i)$ of traveling from city i to city $i+1$ for $i < n$ and from city n to city 1 otherwise, and m items (each has a profit p_i and a weight w_i) distributed in the cities. The availability of items is given by a function $a(i) = \{c_1^i, \ldots, c_{k_i}^i\}$. Additionally, we are given a natural number W (the capacity of the knapsack), a non-negative real number R (the rent ratio) and two positive real numbers $v_{min} \leq v_{max}$ corresponding to the minimal and the maximal speed of the traveler. The task is to find a function assigning to an item i a city $x_i \in a(i) \cup \{0\}$ where item i is picked (0 refers to not picking the item at all) such that the collective total weight of items does not exceed W, and the following is maximized:

$$\begin{aligned} P &= \sum_{i=1}^{m} p_i [x_i \neq 0] - R \sum_{i=1}^{n} t_i \\ t_i &= \frac{d(i)}{v_{max} - W_i \frac{v_{max} - v_{min}}{W}} \end{aligned} \tag{5}$$

where W_i is the total weight of the picked items from cities $\{2, \ldots, i\}$ (we assume that items from city 1 are picked at the end of the tour).

Our decomposition of TTP-PROB is based on the above approach with two exceptions: in a modified KRP the availability function is determined by a scenario, and in a modified TSKP there is given an initial partial tour that has to be extended to the full tour. In fact both of the components of TTP-PROB are easily reducible to the respective components of TTP.

The negotiation protocol between these components is presented as Alg. 2. Given an instance of TTP-PROB, CoSolver iteratively extends partial tour Π and partial picking of items K by the following negotiations. We start by creating an instance of KRP that consists of all items from all nodes and distances according to scenario c and $d(k)$ equal zero. After finding a solution K'' for this instance, we create an instance of TSKP by assigning to each city a weight equal to the total weights of items picked at the city by KRP and providing a partial tour Π. A solution for TTP at the initial step consists of a pair K'', Π'', where Π'' is the route found as a solution to the instance of TSKP. Then the profit P' of the solution is calculated. If profit P' is better than the best profit P that has been found so far, the process repeats with distances between nodes adjusted along tour Π''.

We may obtain various variants of CoSolver algorithms by plugging various KRP and TSKP components in the negotiation protocol. It was suggested in [20] that in most cases the following approach works the best. The KRP component is heuristically reduced to the classical KP [20] and then solved by a weighted greedy approach as explained in [18]. The TSKP component is heuristically reduced to the classical TSP [20] and then solved by the state-of-the-art heuristics (e.g. Concorde [8], which bases on Chained Lin-Kernighan method [2]).

Algorithm 2: CoSolver for TTP-PROB

```
Π ← ∅, K ← ∅
for i ← 1 to n do
    C ← prepare a random scenario
    d_k ← 0, W_k ← 0, P ← −∞
    for r ← 1 to MaxIter do
        K'' ← solve KRP with p_k, w_k, d_k and parameter
              W_k on scenario C and initial picking K
        W_k ← Σ_{i∈K''} w_i[0 < x_i ≤ k]
        Π'' ← solve TSKP with W_k, d and initial path Π
        P'' ← calcObj(K'', Π'')
        if P'' > P' then
            P' ← P''
            Π' ← Π''
            K' ← K''
            d_k ← d(Π_k, Π_{k+1})
        else break
    extend Π with the next node m according to Π'
    extend K with items from K' at node m
return Π, K
```

6. BENCHMARKS

In order to compare performance of the proposed algorithms, we have prepared a generic framework for generating classes of TTP-PROB instances. Each class is composed of three independent components: *meta*, *graph*, *items*, *associations* and *scenarios* that are explained in the below. Depending on parameters configuration of these components one is able to create separate classes of TTP-PROB instances. All instances used in our benchmarks are publicly available at the website [19].

1. Meta. The *meta* component describes parameters of the thief — i.e. knapsack-capacity, rent rate, minimal velocity and maximal velocity. By tunning these parameters we can alter relative importance of TSP or KP components in a TTP. As an example, a smaller value for the rent rate (w.r.t. the value of items) makes the contribution of the TSP component minimal. In an extreme case, the TSP component is completely ignored if the rent rate is zero. Also, by increasing the value of the rent rate the value of items (total profit) becomes less important in the final objective value, consequently, the solution for the KP component have smaller impact on the total objective. This is also the case for the value of $v_{max} - v_{min}$. As an example, lets assume that v_{min} is a constant (say, 0.1). If the value of $v_{max} - v_{min}$ becomes zero, the value of velocity, calculated by $v_{max} - (v_{max} - v_{min})\frac{W_c}{W}$, where W_c is the total weight of the picked items, becomes a constant for any value of available capacity (calculated by $\frac{W_c}{W}$). Thus, picking any item has no impact on the speed of

the travel, and hence, one can completely decompose the TTP instance to two independent sub-problems (TSP component and KP-component) and solve them separately through maximizing the profit in the KP component and minimize the tour distance.

2. Graph. The *graph* component describes the TSP aspect of TTP-PROB — i.e. it describes the graph of cities and distances between cities. To build a competitive set of benchmarks, we decided to use a well-known public database of symmetric and asymmetric TSP instances. The database is available at: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

3. Items. The *items* component describes the KP aspect of TTP-PROB — i.e. it describes items as pairs weight-profit. We used a two-step procedure to build our benchmarks. In the first step we generated sets of items according to the formula:

$$\begin{aligned} w \in &\quad [w_{\min}, w_{\max}] \\ p = &\quad w\delta(1+\rho\gamma) \end{aligned} \tag{6}$$

where the weight of an item is uniformly sampled from the interval $[w_{\min}, w_{\max}]$, δ is the scaling factor, ρ is a normal random variable with mean 0 and standard deviation 1 and γ is a deviation parameter. In the second step each set of items was rescaled according to the role of the node that the items would be assigned to by the *associations* component. We used a parabolic rescaling along "the best path in a graph" — i.e. if we are given n sets of items, then the value p of each item in k-th set is rescaled according to the formula:

$$p' = a + (b-a)\left(\frac{2k}{n}\right)^2 \tag{7}$$

where a and b are parameters such that items from $\frac{n}{2}$-th set are rescaled by a, and items from the first and the last sets are rescaled by b.

4. Associations. The *associations* component links items with nodes in a graph — i.e. it describes which items are available in which nodes at which probabilities. To make the connection between KP and TSP components highly non-trivial, we built the set of associations in the following way: for a given graph we found the best TSP route and according to it ordered the nodes; the k-th node was assigned items from the k-th set of items generated by parabolic rescaling. Therefore, statistically the most profitable items were assigned to the cities that are near to the middle of the optimal route. Probabilities of items were chosen uniformly from the interval $[0, 1]$.

5. Scenarios. The *scenarios* component for a given instance says which items turn to be available at which cities when the thief enters them. Scenarios are crucial to guarantee that both of our algorithms work in the same environment. The scenarios are generated uniformly according to the probabilities of items. Additionally two scenarios are attached: the worst-case scenario (there are no available items) and the best-case scenario (all items are available).

Table 1: Typical performance of Alg. 1 and 2.

Typical	few unif		few parab		uniform		parabolic		KP based	
	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS
att48	387	283	-126	-681	3940	1395	-3595	-11920	1585	696
bayg29	98	104⋆	-67	-119	1967	1240	-1325	-2675	1048	413
bays29	97	95	-62	-116	1979	1197	-1320	-3176	1049	325
berl52	145	96	-49	-331	3374	1504	-3131	-10809	1645	569
br17	88	121⋆	36	9	754	570	-499	-861	416	181
braz58	434	289	-129	-741	2333	841	-1558	-6975	1186	387
burm14	126	84	-40	-43	1497	1261	-1095	-1358	773	253
dant42	232	132	-70	-516	4036	1385	-2992	-9567	1850	448
eil51	153	97	-42	-275	1430	685	-959	-3790	713	374
eil76	579	315	-185	-1512	3411	945	-3425	-14267	1425	563
fri26	29	59⋆	-82	-121	1818	1006	-769	-2134	1098	293
ft53	126	86	-162	-424	3806	1603	-3224	-10123	1689	574
ft70	485	514⋆	-450	-755	3444	2486	-2958	-5685	1903	572
ftv33	51	55⋆	-64	-132	2127	1147	-1763	-4885	951	421
ftv35	234	170	-241	-515	2110	1297	-1534	-4346	978	475
ftv38	125	87	-16	-70	1763	1099	-1774	-4334	678	227
ftv44	294	254	-219	-686	2566	1157	-1643	-6037	1106	457
ftv47	377	288	-133	-585	3797	1384	-3634	-10413	1840	535
ftv55	388	153	-269	-1176	4306	1134	-3398	-13441	1878	836
ftv64	142	138	-232	-856	4011	892	-3545	-15408	1734	687
ftv70	590	223	-323	-1607	2563	569	-2655	-11633	1149	350
gr17	148	120	38	15	854	517	-471	-800	398	237
gr21	160	109	-28	-95	2166	1282	-1484	-2598	985	418
gr24	199	185	35	-66	1730	1325	-1319	-3236	891	349
gr48	364	284	-235	-633	3918	1689	-3543	-11448	1586	501
hk48	352	287	-142	-661	3837	1646	-3555	-11703	1586	492
p43	163	131	-166	-198	1577	1510	-1312	-2110	678	332
pr76	490	274	-191	-1658	3610	725	-3394	-16716	1523	503
ry48p	386	286	-129	-553	3950	1839	-3590	-9573	1851	475
swiss42	165	116	-71	-442	3992	1730	-3133	-9276	1850	666
ulyss16	154	130	-9	-39	1539	1135	-670	-1163	913	317
ulyss22	31	73⋆	-93	-134	1788	1148	-1006	-1824	917	399

7. EXPERIMENTAL RESULTS

We have prepared a set of benchmarks for TTP-PROB using components described in Section 6. It is out of the scope of this paper to compare our algorithms on all prepared instances, therefore in the below we discuss only the most interesting cases:

- (uniform items) $v_{\min} = 1.0, v_{\max} = 2.0$; the knapsack capacity W is set to the total weight of all items divided by 4; the rent rate R is set to the value that balances the average time along the best path and the average total profit of items from a full knapsack; the total number of items is between $2n$ and $8n$, where n is the number of cities in the graph; the minimal and maximal weight of an item are: $w_{\min} = 1, w_{\max} = 100$; the scaling parameter δ is 1.0; the deviation parameter γ is set to 0.4 (see: Eq. (6)); and $a = b = 1$ (see: Eq. (7)),
- (parabolic items) like uniform items, but parabolic parameters are: $a = 8, b = 1$,
- (uniform few items) like uniform items, but the total number of items is between $\frac{n}{8}$ and $\frac{n}{2}$,
- (parabolic few items) like parabolic items, but the total number of items is between $\frac{n}{8}$ and $\frac{n}{2}$,
- (KP Centric) like parabolic items, but the rent rate R is 2^{10} times smaller.

The full set of benchmarks together with the results is publicly available at the website [19].

We limited the time that can be spent by an algorithm on a single scenario to one minute, and focused on the quality of the solutions (i.e. the performance of an algorithm). Since the performance of an algorithm on a given instance depends on a particular scenario (i.e. which items appear when the thief enters a city), to compare various aspects of algorithms we used OWA operators. Results obtained by CoSolver and

Table 2: Average performance of Alg. 1 and 2.

Average	few unif		few parab		uniform		parabolic		KP based	
	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS
att48	342	250	-143	-687	3743	1035	-3638	-11831	1533	553
bayg29	95	84	-67	-118	1880	1022	-1298	-2836	1015	351
bays29	94	90	-61	-113	1886	987	-1293	-3218	1017	298
berl52	145	105	-64	-344	3271	1086	-3200	-10601	1618	494
br17	85	113⋆	21	-9	737	525	-503	-867	405	167
braz58	399	230	-143	-782	2280	591	-1613	-7003	1123	341
burma14	105	92	-40	-43	1433	1043	-1124	-1343	765	252
dant42	224	122	-83	-550	3804	1095	-3018	-9959	1783	412
eil51	152	78	-58	-315	1362	561	-1020	-3795	677	299
eil76	537	230	-240	-1506	3296	644	-3459	-14594	1385	450
fri26	40	55⋆	-82	-122	1704	859	-769	-2231	1075	308
ft53	129	98	-162	-441	3423	1250	-3292	-10233	1539	551
ft70	464	441	-471	-790	3309	2180	-3023	-5722	1777	510
ftv33	68	51	-65	-144	1991	983	-1805	-4721	953	398
ftv35	235	165	-232	-514	2040	1115	-1606	-4350	914	402
ftv38	105	83	-22	-77	1705	924	-1776	-4187	692	228
ftv44	263	217	-217	-673	2434	948	-1684	-6008	978	397
ftv47	323	247	-149	-625	3635	1098	-3679	-11003	1752	477
ftv55	382	120	-258	-1177	4062	768	-3458	-13576	1864	638
ftv64	150	109	-230	-848	3777	606	-3584	-15227	1551	517
ftv70	556	184	-325	-1659	2459	274	-2722	-11984	1099	333
gr17	131	118	23	-1	807	452	-476	-809	394	186
gr21	159	97	-28	-95	2103	1140	-1548	-2672	980	371
gr24	182	152	35	-66	1675	1199	-1355	-3224	881	290
gr48	319	239	-227	-637	3746	1332	-3591	-11446	1534	513
hk48	306	236	-156	-703	3670	1153	-3601	-11951	1534	449
p43	154	125	-174	-232	1605	1258	-1412	-1985	664	272
pr76	493	209	-245	-1604	3427	169	-3430	-16871	1450	435
ry48p	346	237	-145	-565	3751	1466	-3640	-9745	1733	470
swiss42	162	99	-84	-470	3767	1390	-3173	-9646	1783	589
ulyss16	140	126	-9	-47	1468	1001	-671	-1203	913	280
ulyss22	43	68⋆	-93	-130	1725	1020	-1024	-1901	843	342

Table 3: Risk-avoiding performance of Alg. 1 and 2.

Risk-avoiding	few unif		few parab		uniform		parabolic		KP based	
	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS
att48	8	-99	-230	-694	301	-1478	-4029	-11797	472	-140
bayg29	17	0	-67	-125	471	-327	-1629	-3717	552	6
bays29	17	0	-67	-109	472	-424	-1628	-3709	552	-14
berl52	40	-34	-128	-430	110	-1694	-3794	-10706	371	-86
br17	12	5	-41	-83	260	-79	-596	-1098	303	6
braz58	67	-92	-214	-834	232	-1139	-2083	-7240	351	-66
burma14	16	14	-40	-42	378	-160	-1238	-1504	479	18
dant42	94	-50	-123	-588	450	-1679	-3681	-11497	666	-154
eil51	22	-50	-122	-409	102	-565	-1375	-4196	192	-35
eil76	97	-210	-371	-1685	20	-2178	-4036	-15236	218	-160
fri26	14	-9	-82	-153	535	-291	-1011	-2696	654	9
ft53	21	-39	-162	-492	210	-1256	-3738	-9737	368	-105
ft70	68	-23	-497	-880	271	-499	-3547	-6074	503	9
ftv33	16	-11	-70	-183	242	-694	-2026	-4852	363	-23
ftv35	51	-36	-201	-547	128	-593	-1975	-5193	195	-30
ftv38	15	4	-32	-83	270	-486	-1868	-4026	348	-34
ftv44	43	-67	-238	-664	233	-798	-2124	-6232	274	-42
ftv47	43	-70	-231	-695	281	-1640	-4096	-11994	537	-142
ftv55	131	-142	-268	-1409	308	-2415	-4078	-15028	525	-150
ftv64	27	-93	-231	-772	136	-2174	-4192	-15886	270	-164
ftv70	140	-195	-381	-1627	-45	-1769	-3220	-12297	154	-149
gr17	16	11	-41	-61	276	-80	-592	-811	299	11
gr21	87	1	-28	-109	540	-366	-1949	-3427	634	2
gr24	132	-6	35	-86	280	-249	-1649	-3355	367	1
gr48	40	-75	-238	-745	317	-1521	-4014	-12216	472	-147
hk48	40	-68	-232	-831	293	-1805	-4020	-13206	472	-170
p43	29	-18	-197	-221	142	-90	-1657	-1885	185	-16
pr76	93	-248	-372	-1565	43	-2312	-4030	-16248	246	-194
ry48p	44	-68	-230	-634	296	-1241	-4086	-11350	513	-105
swiss42	92	-63	-123	-498	465	-1458	-3716	-10990	666	-96
ulyss16	57	10	-9	-59	631	-28	-764	-1278	721	19
ulyss22	15	4	-93	-128	385	-225	-1204	-2028	441	16

MCTS algorithms have been compared against each other according to four types of behaviours as described in Section 4: typical (Table 1), average (Table 2), cubical (i.e. of order 3) risk-avoiding (Table 3) and cubical risk-seeking (Table 4). Each algorithm was run five times on a single scenario: we removed the best and the worst solutions and took the average of the remaining three solutions as the result (rounded to the nearest integer). It should be noted that the complexity of TTP (let alone its probabilistic version PTTP) makes the problem intractable by any exact algorithm on instances of any reasonable size (as explained in [20], there is no polynomial constant-factor approximation algorithm for TTP unless P = NP), therefore we were not able to compare the results obtained by our algorithms against the best solutions attainable with the perfect knowledge.

The results presented in Tables 1, 2, 3 and 4 show that CoSolver with integrated problem-specific components of TTP-PROB outperforms MCTS on most instances (we marked by "⋆" the only entries where MCTS was better than CoSolver). In fact, CoSolver performed strictly better on all instances excluding "uniform few items", in which cases MCTS was able to outperform CoSolver 6 times on typical behaviour, 3 times on average behaviour and 13 times on risk-seeking behaviour (out of 32 times in each case). This confirms our initial hypothesis that when faced with a new real-world problem, instead of building an algorithm from scratch, it may be more reasonable to use existing algorithms as building blocks for the problem and focus the work on negotiation protocols between the components.

Notice also, that it is not generally true that MCTS behaves in a more risk-seeking manner than CoSolver, as it may be suggested by the above analysis. By computing the average gaps from the results produced by CoSolver to the results produced by MCTS according to different behaviours, one may actually discover the opposite: MCTS is more risk-avoiding and less risk-seeking than CoSolver. The average gaps are: 1501 for the typical behaviour, 1538 for the average behaviour, 1868 for the risk-seeking behaviour, and 1435 for the risk-avoiding behaviour. This may be explained by the fact that the MCTS-based algorithm, as purely stochastic, performed in a more uniform way. The seemingly contradictory observation that MCTS was not able to outperform CoSolver on any instance according to the risk-avoiding behaviour, but was better than CoSolver on 13 instances from "uniform few items" according to the risk-seeking behaviour, follows from the fact that when there are very few almost equally good items, then CoSolver as being optimistic about further possibilities may unnecessarily waste good items that are available at a given moment and end up with nothing.

8. CONCLUSIONS AND FURTHER WORK

Our long-term goal is to provide a broad new methodology for integration of real-world problems, progressing from simpler couplings of silos and sequences, to heterogeneous highly connected models.

In this paper we addressed one additional aspect of real-world problems: uncertain environment. We proposed a new model problem that extends the Traveling Thief Problem with the aspect of uncertainty and showed how both the decomposition-based approach and Monte-Carlo Tree Search can be adapted to this new setting. We also prepared a public database of instances for this new problem.

Our benchmark confirms that CoSolver outperforms classical MCTS-based algorithms. Moreover, classical implementation of MCTS algorithm exhibits more risk-avoiding and less risk-seeking behaviour than CoSolver. It is an interesting question whether we can adjust this behaviour by using OWA-driven approach in the exploration phase in MCTS. We leave it for future work.

In future work we will also be interested in extending our model problem with additional aspects that may be found in real-world systems, and in developing new decomposition-based methodologies for such extensions.

Table 4: Risk-seeking performance of Alg. 1 and 2.

Risk-seeking	few unif		few parab		uniform		parabolic		KP based	
	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS	CoSo	MCTS
att48	493	373	-122	-658	6081	1218	-3646	-10537	2483	761
bayg29	116	33	-67	-110	2804	1061	-1341	-2712	1271	329
bays29	115	79	-48	-106	2807	1208	-1337	-3146	1272	338
berl52	92	146⋆	-47	-336	5699	1262	-3153	-10468	2512	477
br17	120	150⋆	38	13	1093	696	-491	-795	463	219
braz58	475	236	-147	-747	3652	842	-1690	-7053	1555	310
burm14	89	127⋆	-40	-43	2185	1132	-1130	-1351	901	310
dant42	213	208	-81	-581	5862	1643	-2945	-8971	2655	497
eil51	213	98	-74	-361	2354	645	-1057	-3644	934	240
eil76	471	229	-267	-1491	5675	898	-3239	-15176	2214	398
fri26	36	73⋆	-82	-98	2408	1069	-699	-1900	1282	313
ft53	90	155⋆	-162	-463	5344	1306	-3229	-10135	2250	827
ft70	380	483⋆	-518	-841	5436	2609	-3101	-5935	2555	581
ftv33	89	65	-63	-134	3362	1719	-1812	-4358	1468	508
ftv35	327	166	-209	-510	3336	1272	-1711	-4529	1320	522
ftv38	88	117⋆	-33	-102	2857	1074	-1786	-4330	946	260
ftv44	248	291⋆	-227	-623	3681	1408	-1649	-5957	1479	406
ftv47	321	294	-130	-605	6036	1474	-3641	-12116	2495	683
ftv55	420	171	-235	-1080	6282	1534	-3418	-13144	2708	535
ftv64	178	86	-228	-845	6180	995	-3578	-15349	2480	490
ftv70	484	272	-316	-1605	4639	350	-2711	-12497	1730	550
gr17	134	163⋆	40	18	1105	496	-457	-929	463	240
gr21	194	68	-28	-81	3150	1496	-1539	-2888	1349	400
gr24	202	143	35	-38	2869	1617	-1411	-3737	1193	212
gr48	326	300	-227	-658	6068	1680	-3602	-11242	2484	1030
hk48	335	266	-139	-796	6032	1081	-3613	-10711	2484	729
p43	122	150⋆	-197	-253	2773	1639	-1443	-1783	1025	271
pr76	482	252	-270	-1498	5706	-60	-3208	-16667	2208	763
ry48p	389	307	-125	-557	6085	1702	-3611	-9891	2511	891
swiss42	104	181⋆	-82	-491	5840	1984	-3096	-9248	2655	751
ulyss16	134	164⋆	-9	-62	1942	1117	-749	-1412	1078	360
ulyss22	36	75⋆	-93	-121	2536	1326	-1125	-1918	972	343

9. REFERENCES

[1] B. Abramson. *The Expected-outcome model of two-player games*. Research notes in artificial intelligence. Pitman, London, 1991.

[2] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[3] D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[4] M. R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1037–1044, 2013.

[5] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 421–428, New York, NY, USA, 2014. ACM.

[6] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms i: Representation. *Comput. Ind. Eng.*, 30(4):983–997, Sept. 1996.

[7] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

[8] W. J. Cook. A computer code for tsp. http://www.math.uwaterloo.ca/tsp/concorde.html, 1995.

[9] L. Davis. Job shop scheduling with genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, volume 140. Carnegie-Mellon University Pittsburgh, PA, 1985.

[10] J. Faires and R. Burden. *Numerical Methods*. Number t. 1 in Numerical Methods. Thomson/Brooks/Cole, 2003.

[11] A. Griewank and P. L. Toint. Local convergence analysis for partitioned quasi-newton updates. *Numerische Mathematik*, 39(3):429–448, 1982.

[12] M. Kearns. Experiments in social computation. *Commun. ACM*, 55(10):56–67, Oct. 2012.

[13] X. Li and X. Yao. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1546–1553. IEEE, 2009.

[14] X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *Evolutionary Computation, IEEE Transactions on*, 16(2):210–224, 2012.

[15] Z. Michalewicz. Quo vadis, evolutionary computation? In *Advances in Computational Intelligence*, pages 98–121. Springer, 2012.

[16] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern Heuristics*. Springer New York, 2000.

[17] W. Ogryczak, P. Perny, and P. Weng. On minimizing ordered weighted regrets in multiobjective markov decision processes. In *Proc. ADT 2011*, pages 190–204. Springer LNAI 6992, 2011.

[18] D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[19] M. R. Przybylek. Multihard problems. https://sites.google.com/site/travellingthief.

[20] M. R. Przybylek, Z. Michalewicz, and A. Wierzbicki. *Evolutionary Computation Journal*. MIT Press on Combinatorial Optimization Problems, To appear.

[21] S. Schellenberg, A. Mohais, M. Ibrahimov, N. Wagner, and Z. Michalewicz. In R. Chiong, T. Weise, and Z. Michalewicz, editors, *Variants of Evolutionary Algorithms for Real-World Applications*. Springer, 2012.

[22] V. P. Singh, B. Duquet, M. Léger, and M. Schoenauer. Automatic wave-equation migration velocity inversion using multiobjective evolutionary algorithms. *Geophysics*, 73(5):VE61, 2008.

[23] E.-G. Talbi. *Metaheuristics for Bi-level Optimization*. Springer Publishing Company, Incorporated, 2013.

[24] S. Tesfamariam and R. Sadiq. Probabilistic risk analysis using ordered weighted averaging (owa) operators. *Stoch Environ Res Risk Assess*, 22:1–15, 2006.

[25] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[26] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Oper. Res.*, 40(1):113–125, Jan. 1992.

[27] A. Wierzbicki. *Trust and Fairness in Open, Distributed Systems*. Springer, 2010.

[28] R. Yager. On ordered weighted averaging aggregation in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18:183–190, 1988.

[29] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.