Decomposition Algorithms for a Multi-hard Problem

M. R. Przybylek mrp@pjwstk.edu.pl
Polish-Japanese Academy of Information Technology, Warsaw, Poland

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

Z. Michalewicz* zm@complexica.com
Complexica, Adelaide, Australia

Abstract

Real-world optimization problems have been studied in the past, but the work resulted in approaches tailored to individual problems that could not be easily generalized. The reason for this limitation was the lack of appropriate models for the systematic study of salient aspects of real-world problems. The aim of this paper is to study one of such aspects: multi-hardness. We propose a variety of decomposition-based algorithms for an abstract multi-hard problem and compare them against the most promising heuristics.

Keywords

Traveling Thief Problem, Co-evolution, Metaheuristics, Multi-hard problems, Multi-objective optimization, Non-separable problems, Real-world optimization problems

1 Introduction

Mathematical modeling has been the basis of many natural sciences, as well as operations research, for decades. Yet, even as many advances have been made, over the years the phenomenon of "unreasonable ineffectiveness of mathematics" in computer engineering (see: Gunawardena (1998)), cognitive science (see: Poli (1999)), economics (see: Velupillai (2005)), and biology (see: Borovik (2009)) has been noticed. In Michalewicz (2012) and Michalewicz and Fogel (2000) authors argue that the same phenomenon occurs in real-world optimization. They divided hard optimization problems into two categories: designed and real-world problems. "Designed problems" are "mathematical" — they have simple logical formulation, are surgically precise, and the objective function clearly indicates a better solution out of two potential solutions. This category includes the Traveling Salesman Problem (TSP), Graph Colouring Problem (GCP), Job Shop Scheduling Problem (JSSP), and Knapsack Problem (KP), to name a few. Real-world optimization problems have not been designed by anyone, but occur in real business processes. They usually have very complex formulations. To solve such problems, first we have to build their models, and the quality of the obtained solution will depend on the quality of the model.

*Z. Michalewicz is also with Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland, and the Polish Japanese Academy of Information Technology, Warsaw, Poland.

The level of difficulty of designed and real-world optimization problems differs in practice, even if they may be equivalent from the point of view of complexity theory. In this article, the distinction between these levels of difficulty is made by referring to "single-hard", "double-hard" and more generally "multi-hard" problems. A single-hard problem means a designed problem of high computational complexity. A "multi-hard" problem can be described as a non-trivial combination of "single-hard" problems: solving sub-problems of a multi-hard problem in isolation does not lead to a good solution of the multi-hard problem.

Through a better understanding of multi-hardness, the "ineffectiveness of mathematics" for solving real-world optimization problems may be reduced. Often, in our attempts to reduce the complexity of multi-hard problems, we create models that use known, single-hard problems that are combined by additional interdependencies like joint criteria or joint constraints.

The aim of this article is to develop foundations for solving multi-hard problems. The starting point is a formulation of an abstract double-hard problem, called the Traveling Thief Problem (TTP) in Bonyadi et al. (2013) that is a non-trivial composition of two well-studied classical problems: the Traveling Salesman Problem and the Knapsack Problem.

In this work, TTP is studied further, with the goal of obtaining insights into the difficulty of multi-hard problems in general through an evaluation of algorithms for solving TTP. The goal is to compare known heuristics against algorithms that aim to exploit the structure of a multi-hard problem.

Specialized algorithms have been developed for many (perhaps most) well-known single-hard problems. Unfortunately, such algorithms are often very sensitive to problem modifications, such as new constraints. Moreover, such algorithms do not exist for multi-hard problems, and it would be hard to develop them as multi-hard problems are defined by special combinations of single-hard problems of various types. The way these single-hard problems are combined differs from one multi-hard problem to another, as well.

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. The first candidates would be known meta-heuristics. After all, multi-hard problems are in the same computational complexity class as single-hard problems. However, this approach does not take into account the structure and type of combination of a multi-hard problem.

In our previous work in Bonyadi et al. (2014) we have developed the idea of CoSolver and applied it to the Traveling Thief Problem 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 algorithms solving sub-problems, and then combine the solutions back to obtain a solution to the initial problem.

This paper makes the following contributions. CoSolver is compared against meta-heuristics that we have thought of as most promising for multi-hard problems: a Monte-Carlo Tree Search algorithm and Ant Colony Optimization. The algorithms are also compared to exact solutions for a variety of instances of TTP, differing in difficulty and structure. Further, CoSolver is extended by incorporating heuristics instead of exact solvers for the TSP and KP components of TTP. This extension greatly improves the scalability of CoSolver without compromising quality.

The structure of the paper is as follows. In the next section we discuss related

work. Section 3 formally defines the Traveling Thief Problem. Section 4 introduces the concept of decomposition algorithms for multi-hard problems, the CoSolver algorithm, and the Monte-Carlo Tree Search algorithm for TTP. Section 5 describes the benchmark instances of TTP. Section 6 presents results of experiments with solving benchmarks using proposed algorithms. Section 7 concludes the paper.

2 Related Work

In 2013 The Traveling Thief Problem (TTP) was introduced (see: Bonyadi et al. (2013)) as an example of multi-level optimization problem. The problem was presented as a combination of two well-known sub-problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). The authors showed that optimal solutions for each sub-problem do not guarantee a global solution, because the interdependency between the two problems affects the optimal solution for the whole problem. Although some extensions to the Traveling Salesman Problem were studied before, they consisted of one hard problem (i.e. the core problem, which was usually the Vehicle Routing Problem, see: Braekers et al. (2015), and Toth and Vigo (2001)) equipped with additional constraints, and were solved as single monolithic problems. The variants of the Vehicle Routing Problem that are closest to the Traveling Thief Problem are: the Time Dependent Vehicle Routing Problem (see: Malandraki and Daskin (1992)), where the cost of traveling between the cities varies over time, and the Capacitated Vehicle Routing Problem (see: Ralphs et al. (2003)), where the travelers (vehicles) have additional constraints on the total weight of items that can be carried.

2.1 State of The Art

- As noted in Bonyadi et al. (2013) most of the current research in algorithm design focuses on problems with a single hard component (Traveling Salesman Problem, Knapsack Problem, Job Shop Scheduling Problem (see: Cheng et al. (1996), Davis (1985), and van Laarhoven et al. (1992)), the Vehicle Routing Problem (see: Toth and Vigo (2001)), etc.), whilst most of the real-world problems are multi-hard problems. It has also been shown in the paper that interdependencies between components of multi-hard problems play a chief role in the complexity of these problems. Such interdependencies do not occur in designed, single-hard problems (see, for example: Bonyadi et al. (2014)). In order to present the complexity that results from interdependencies in multi-hard problems, the Traveling Thief Problem was introduced.

- Bonyadi et al. (2014) introduced a new algorithm (called CoSolver) for solving multi-hard problems, which focuses on the aspect of communication and negotiation between partial sub-problems. The main idea behind CoSolver is to decompose TTP into sub-problems, solve the sub-problems separately, with some communication between the algorithms solving them, and then combine such partial solutions back to the overall solution for TTP. The article also proposed a simple heuristics (called Density-Based Heuristic) as a second approach and compared it to CoSolver. This heuristic first generates a solution for the TSP component of a given TTP problem and then for the found TSP route solves the generalized KP problem so that the objective value is maximized. It is worth noting that Density-Based Heuristic ignores all of the interdependencies between the sub-problems. These two algorithms were compared on a series of benchmark instances. The results showed that the CoSolver efficiency was better than Density-Based Heuris-

tic, suggesting that taking into consideration the interdependencies between the sub-problems is beneficial.

- It was argued in Przybylek et al. (2016) that multi-hardness is not the only crucial aspect of real-world optimization problems. Another important characteristic is that real-world problems usually have to operate in an uncertain and a dynamically-changing environment. This observation resulted in a formulation of a probabilistic variant of TTP. The authors also showed how the decomposition-based approach (namely, CoSolver) can be incorporated in this new setting.

- The first attempts to solve multi-hard problems were based on methods for large-scale optimization. Typical methods used in such approaches are: Newton's method and conjugate gradient method (see: Faires and Burden (2003)), the partitioned quasi-Newton method (see: Griewank and Toint (1982)), and linear programming (see: Bertsimas and Tsitsiklis (1997)). The main disadvantage of these methods is, however, their dependency on the algebraic formalization of the problem and the availability of information about the gradients. In case of many of the real-world problems an algebraic formalization is simply impossible. Therefore, the simulation has to be used to get the evaluation of potential solutions, by providing an output value for a given set of input decision values. This kind of optimization (i.e. a black-box optimization) is widely used in mechanical engineering and many other disciplines. In black-box optimization, meta-heuristics such as evolutionary algorithms (EAs) have a considerable advantages over conventional single-point derivative-based methods of optimization. Meta-heuristics do not base on gradient information and are less likely to stuck on local optima due to the use of a population of possible solutions. In addition, the recent advances in meta heuristics show that the cooperative coevolutionary algorithms hold great promise for such problems as shown in Yang et al. (2008), Li and Yao (2009), and Li and Yao (2012). Nonetheless, major challenges remain. Finally, there is also modern research on multi-level optimization, where the optimization problems consist of multiple sub-components that are subject to certain hierarchical relationships (see: Colson et al. (2007), and Talbi (2013)). In such setting, the components that are lower in the hierarchy do not include in the optimization process the solutions of the components that are higher in the hierarchy.

- To solve complex problems human computational potential can be used (see: Kearns (2012)). This is, however, a completely new approach in the context of multi-hard 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 Real-World Inspiration

Real, multi-hard optimization problems are solved in practice every day by human decision makers. A good example of such multi-hard problem is the optimization of the supply chain operations from the mines to the ports (see also: Bonyadi and Michalewicz (2016), and Bonyadi et al. (2016)). These operations include mine planning and scheduling, stockpile management and blending, train scheduling, and port operations — with an overall objective to satisfy customer orders by providing predefined quantity of products by a specified date.

Lets look at the some of these operations in more detail:

1. *Train scheduling*. To operate the trains, there is a railway network, which is usually hired by the mining company so that trains can travel between the mines and the ports. The owner of the railway network sets some restrictions for the operation of trains for each mining company, for example: the number of trains per day for each junction in the network is constant (set by the owner of the railway network) for each mine company. There are a number of self-unloading stations, which are scheduled to unload products from the trains arriving in the port. The mine company schedules and loads trains in the mines of requested material and then sends them to the port, while respecting all constraints (that is, the train scheduling procedure).

2. *Train unloading*. The mine company also has to plan train dumpers to unload the trains and place unloaded products at a port. A port encloses a huge area called the stockyard, several places to berth ships (called the berth) and a waiting area for ships. The stockyard contains certain stockpiles, which are storages of individual products (mixing products in stockpiles is not allowed) and are limited by some capacities.

3. *Ship scheduling*. Ships arriving in ports (time of arrival is often approximate, due to weather conditions) have to wait in the waiting area until the port manager assigns them to a particular berth, where they take specific products to be delivered to the customers. Ships are subject to the penalty costs, called the demurrage — the penalty is applied for each unit of time while the ship is waiting in the port of its arrival. The mining company's goal is to plan the ships and fill them with the requested products, so that the total demurrage consumed by all ships is limited to a minimum.

4. *Loading ships*. There are several ship loaders, which are assigned to each berthed ship to load it with requested products. The ship loaders take products from appropriate stockpiles and load them into ships. It should be noted that different ships may have different requirements for the products, and each product can be taken from a different stockpile, so that the scheduling various ship loaders and choosing different stockpiles in order to meet the demand of the ships may result in different amount of time to finish the loading. It is the task of the owner of the mine to ensure sufficient quantities of each type of products in the stockyard until the ships arrive.

Each of the above-mentioned procedures (train scheduling, train unloading, ship scheduling, ship loading) is a component of the optimization problem. Of course, each of these components is a problem on its own, which is difficult to solve. In addition to the complexity of the components, solving the components in isolation does not lead to an overall solution to the whole problem. As an example, the optimal solution to the problem of train scheduling (carrying as much materials as possible from mines to ports) may result in an insufficient amount of available landfill capacity or even the lack of suitable products for ships that will arrive on schedule. Also, the best plan for dumping products from the trains and keeping them in the stockyard can lead to poor quality of the plan for ship loaders, which would have to move too many times to load the ship.

While TTP is an abstract model of a multi-hard problem, it is also inspired by a real multi-hard problem of optimizing supply-chain operations of a mining company. The KP component of TTP models train loading, while the TSP component models

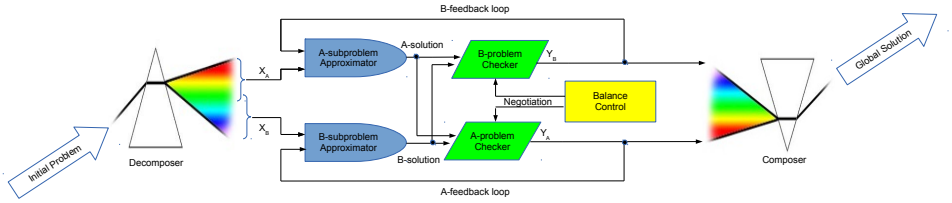

Figure 1: Decomposition of a problem on two sub-problems: A and B.

scheduling a train that has to visit several mines. It is clear from this analogy how TTP could be extended to create new multi-hard problems (possibly with more than two components).

However, this complex, real multi-hard problem is solved by the mining corporation, applying two basic approaches: specialization and collaboration or negotiation (see: Derrouiche et al. (2008)). Teams of decision makers work on each component of the problem, such as scheduling trains and ships, separately. These teams are specialized and experienced in solving their particular problem. This kind of a unit in a corporation is also referred to as a "silo", since it is responsible only for a selected part of operations and does not need deal with other issues. Silos can collaborate and negotiate with each other: a solution proposed by the silo should be consulted with solutions of other silos, since independent solutions frequently interfere or even disturb one another. This collaboration is usually crucial to the success of the management of the whole supply-chain, and hence it is usually the responsibility of higher-level management to carry out or oversee negotiations among silos.

The concept of decomposition algorithms and CoSolver is inspired by this social or managerial solution of real multi-hard problems. Decomposition algorithms use solvers of sub-problems of the multi-hard problem instead of silos of human decision makers. Moreover, a decomposition algorithm needs a method of "negotiation" of the solutions found by its sub-problem solvers. This negotiation method can be crucial to the algorithm's success. The general idea is shown on Fig 1. The initial problem is decomposed onto two, possibly overlapping, sub-problems: X_A and X_B. Each of the sub-problems X_A, X_B (optionally with some knowledge about the solution Y_B, Y_A to the other sub-problem from the previous iteration) is passed to a domain-specific solver giving a partial solution *A-solution* (*B-solution*). Then, the negotiation protocol starts modifying *A-solution* to respect *B-solution* and modifying *B-solution* to respect *A-solution*. Finally, the solutions are composed together to obtain a solution to the initial problem.

4 TTP: a Model Multi-hard Problem

In this section we provide a formal definition of the Traveling Thief Problem (see: Bonyadi et al. (2013)). Given:

- a weighted graph $\mathcal{G} = \langle V, E \rangle$, whose nodes $m, n \in V$ are called cities, and whose edges $m \overset{d(m,n)}{\longrightarrow} n \in E$ from m to n are called distances

- an initial city $s \in V$
- a list of pairs of natural numbers $\langle w_i, p_i \rangle_{1 \leq i \leq I}$, called the list of items; each item $\langle w_i, p_i \rangle$ has its weight w_i and its profit p_i
- a relation a, called the availability of the items, between the cities V and a set $\{1, 2, \ldots, I\}$; the i-th item is available in city $n \in V$ iff $a(n, i)$ is satisfied
- a natural number C, called the capacity of the knapsack
- a real number $R \geq 0$, called the rent ratio
- two positive real numbers $v_{min} \leq v_{max}$, called the minimal and maximal speed

the Traveling Thief Problem (TTP) asks what is the most profitable picking plan on the best route that starts and ends in the initial city s and visits each other city exactly once. In more details, let:

- $\pi = \langle \pi_1, \pi_2, \ldots, \pi_N, \pi_{N+1} \rangle$ be a Hamiltonian cycle in $\mathcal{G}$ such that $\pi_1 = \pi_{N+1} = s$ is the starting node
- $\sigma \colon \{1, 2, \ldots, I\} \rightarrow V$ be a *partial* function ("the picking plan") such that every item i which belongs to the domain of σ is available in city $\sigma(i)$ (that is, $i \in dom(\sigma) \Rightarrow a(i, \sigma(i))$) and the capacity of the knapsack in never exceeded (that is, $\sum_{i \in dom(\sigma)} w_i \leq C$)

the profit of the traveler is defined as:

$$P = \sum_{i \in dom(\sigma)} p_i - R \sum_{n=1}^{|V|} t(\pi_n, \pi_{n+1}) \tag{1}$$

where $t(\pi_n, \pi_{n+1})$ is the time to travel from π_n to π_{n+1} assuming the cumulative weight of all picked items by city π_n on cycle π is $W(n)$. This is given by the formula:

$$\begin{aligned} t(\pi_n, \pi_{n+1}) &= \frac{d(\pi_n, \pi_{n+1})}{v_{max} - W(n) \frac{v_{max} - v_{min}}{C}} \\ W(n) &= \sum_{i \in dom(\sigma) \cap \{i \colon \exists\limits_{m \leq n} a(i, \pi_m)\}} w_i \end{aligned} \tag{2}$$

We shall note that, if more items are picked while traveling through cities, the value $t(\pi_n, \pi_{n+1})$ will grow, which in turn will cause the reduction of the total profit (see the value P in Eq. 1). Likewise, by choosing better Hamiltonian cycles in terms of the total distance, some possibly high quality items (items which have a high profit) might only be available at the beginning of the cycle and, hence, by picking those items, the travel time will increase (items will be carried for a longer time), which in turn will also cause the reduction the total profit. This shows that the interdependency between the two problems influences the optimum solutions for the whole problem. Also, solving each component in isolation does not necessarily lead to the optimum of the problem (see: Bonyadi et al. (2013)).

5 Algorithms

In our previous work (see: Bonyadi et al. (2014)) we have developed the idea of a decomposition-based algorithm and applied it to TTP obtaining some promising results (see also Section 7). Instances of TTP were decomposed into two components[1]: TSKP — the Traveling Salesman with Knapsack Problem and KRP — the Knapsack on the Route Problem. In this section we develop two additional variants of this algorithm: one using a heuristic approach to solve the TSKP component and exact solver for KRP component and another using heuristics for both components. We also describe two heuristic solvers: one which is based on Monte-Carlo Tree Search and another which is based on Ant Colony Optimization. In Section 6 we compare these algorithms against each other.

5.1 Decomposition Algorithms

We have identified the following sub-problems of TTP — one corresponding to a generalization of TSP, which we shall call the Traveling Salesman with Knapsack Problem (TSKP), and another corresponding to a generalization of KP, which we shall call the Knapsack on the Route Problem (KRP).

Let us start with the definition of TSKP. Given:

- a weighted graph $\mathcal{G} = \langle V, E \rangle$, whose nodes $m, n \in V$ are called cities, and whose edges $m \overset{d(m,n)}{\longrightarrow} n \in E$ from m to n are called distances
- an initial city $s \in V$
- a function $\omega \colon V \to \mathcal{R}$ assigning to every node $n \in V$ a non-negative real number $\omega(n)$, which may be thought of as the total weight of items picked at city n
- a natural number C, called the capacity of the knapsack
- a real number $R \geq 0$, called the rent ratio
- two positive real numbers $v_{min} \leq v_{max}$, called the minimal and maximal speed

the Traveling Salesman with Knapsack Problem asks what Hamiltonian cycle π that starts and ends in the initial city s minimizes the following function:

$$T = R \sum_{n=1}^{|V|} t(\pi_n, \pi_{n+1}) \tag{3}$$

where:

$$\begin{aligned} t(\pi_n, \pi_{n+1}) &= \frac{d(\pi_n, \pi_{n+1})}{v_{max} - W(n)\frac{v_{max} - v_{min}}{C}} \\ W(n) &= \sum_{1 \leq i \leq n} \omega(\pi_i) \end{aligned} \tag{4}$$

(compare the above functions with Eq. 1 and Eq. 2 from Section 2). Note that in this formulation of the problem, constants R and C are excessive — we keep them for convenience only.

The Knapsack on the Route Problem is the counterpart of the above problem. Given:

[1] See Section 5.1 for definitions

- a set $V = \langle 1, 2, \ldots, N \rangle$, whose elements are called cities,
- a function $\delta \colon V \to \mathcal{R}$ that assigns to each city n a non-negative real number $\delta(n)$, which may be thought of as the distance from city n to the "next" city on some route
- a list of pairs of natural numbers $\langle w_i, p_i \rangle_{1 \leq i \leq I}$, called the list of items; each item $\langle w_i, p_i \rangle$ has its weight w_i and its profit p_i
- a relation a, called the availability of the items, between the cities V and a set $\{1, 2, \ldots, I\}$; the i-th item is available in city $n \in V$ iff $a(n, i)$ is satisfied
- a natural number C, called the capacity of the knapsack
- a real number $R \geq 0$, called the rent ratio
- two positive real numbers $v_{min} \leq v_{max}$, called the minimal and maximal speed

Knapsack on the Route Problem asks what picking plan $\sigma \colon \{1, 2, \ldots, I\} \to V$ maximizes the following function:

$$P = \sum_{i \in dom(\sigma)} p_i - R \sum_{n=1}^{N} t(i) \tag{5}$$

where:

$$\begin{aligned} t(n) &= \frac{\delta(n)}{v_{max} - W(n) \frac{v_{max} - v_{min}}{C}} \\ W(n) &= \sum_{i \in dom(\sigma) \cap \{i \colon \exists\limits_{m \leq n} a(i,m)\}} w_i \end{aligned} \tag{6}$$

(also compare the above functions with Eq. 1 and Eq. 2 from Section 2).

Observe that our decomposition preserves the relative difficulties of the original components. First of all, because there is a trivial gap-preserving reduction from TSP to TSKP, we obtain the following theorem.

Theorem 1. *There is no polynomial constant-factor approximation algorithm for the Traveling Salesman with Knapsack Problem unless $P = NP$.*

On the other hand, in this section we construct an algorithm for KRP that is polynomial under unary encoding of profits of items (Algorithm 2), which may be turned into a fully polynomial approximation scheme for KRP in the usual way.

Theorem 2. *There is a fully polynomial approximation scheme for the Knapsack on the Route Problem.*

Therefore TSP is computationally equivalent to TSKP and KP is computationally equivalent to KRP.

At this point, one may wonder why we have identified TSKP and KRP subcomponents of TTP, instead of the obvious TSP and KP. As mentioned earlier, two key factors of decomposition-based approach are:

- identification of subcomponents of the problem
- development of a communication protocol for the subcomponents

Algorithm 1: CoSolver

1: $\delta(k) \leftarrow 0$
2: $P^* \leftarrow -\infty$
3: **for** $r \leftarrow 1$ **to** $MaxIter$ **do**
4: $\quad\sigma \leftarrow$ solve KRP with $\langle p_i, w_i \rangle_{i \in I}, \delta$ and parameters C, R, v_{min}, v_{max}
5: $\quad\omega(k) \leftarrow \sum_{i \in dom(\sigma) \wedge a(i,\pi_k)\}} w_i$
6: $\quad\pi \leftarrow$ solve TSKP with ω and parameters d, R, v_{min}, v_{max}
7: $\quad P \leftarrow profit(\pi, \sigma)$
8: $\quad$**if** $P > P^*$ **then**
9: $\quad\quad P^* \leftarrow P$
10: $\quad\quad \pi^* \leftarrow \pi$
11: $\quad\quad \sigma^* \leftarrow \sigma$
12: $\quad\quad \delta(k) \leftarrow d(\pi_k, \pi_{k+1})$
13: $\quad$**else**
14: $\quad\quad$ break
15: **return** σ^*, π^*

These factors are, of course, not completely independent of each other and there are very many important aspects that we have to take into consideration when making such choices:

- there should be efficient approximation algorithms for subcomponents
- the algorithms for subcomponents should be "stable", by what we mean that, whenever possible, similar instances of the problem should lead to similar solutions
- subcomponents have to be chosen is such a way that makes it possible to develop an effective and efficient negotiation protocol
- a good solution to the problem has to be found in a possibly small number of executions of approximation algorithms for subcomponents
- the computational overhead of the communication protocol should be reasonably small

Having the above in mind, we can now better understand our choice of subcomponents of TTP. One could naively think that since TTP has been designed as a generalisation of both TSP and KP problems, the natural choices for subcomponents are exactly TSP and KP. Nonetheless, the highly non-linear interdependencies between TSP and KP parts of TTP, make it difficult to develop efficient and effective negotiation protocol for them.

The negotiation protocol between TSKP and KRP components is presented as Algorithm 1.

Given an instance of TTP, CoSolver starts by creating an instance of KRP that consists of all items of TTP, and distances $\delta(k)$ equal zero. After finding a solution σ for this instance, it creates an instance of TSKP by assigning to each city a weight equal to the total weights of items picked at the city according to σ. A solution for TTP at the initial step consists of a pair σ, 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

Algorithm 2: Pseudo-polynomial solver for KRP

$P[1]$ stores initial profits; if not supplied to the procedure $P[1][w] \leftarrow 0$
for $n \leftarrow 2$ **to** $N + 1$ **do**
  $P[n] \leftarrow$ *solve KP with initial profits P[n-1] and items from city n*
  for $w \leftarrow 0$ **to** C **do**
    $t \leftarrow \delta(n-1)(\frac{1}{v_{max}} - \frac{1}{v_{max} - w\frac{v_{max}-v_{min}}{C}})$
    $P[n][w] \leftarrow P[n][w] - Rt$
$m = 0$
for $w \leftarrow 0$ **to** C **do**
  if $P[N+1][w] > P[N+1][m]$ **then**
    $m \leftarrow w$
track down the structures to find which items σ correspond to the optimal real profit $P[N+1][m]$ with total weight m
return σ

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.

We have implemented the following algorithms for KRP component.

1. Exact solver for KRP (Algorithm 2). The algorithm runs in time and space polynomial under unary encoding of profits of items. It inductively builds a two-dimensional array P, such that $P[n][w]$ stores the real profit that can be obtained by transporting items of total weight w through cities up to n. The initial values $P[1][w]$ are set to zero for every $0 \leq w \leq C$. Assuming that we have computed $P[n-1][w]$, the values $P[n][w]$ can be obtained by using the usual dynamic-programming routine for the Knapsack Problem (Algorithm 3) on items that are available at city n minus the difference in costs between traveling from city $n-1$ to city n carrying the empty knapsack and carrying a knapsack that weights w.

2. Heuristic reduction from KRP to KP and exact solver for KP.

   Let $W(i)$ denote the total weight of items picked at cities $\{1, 2, \ldots, N\}$ according to some picking plan. We create an instance of KP with "relaxed profits" in the following way:

   $$\overline{p}_i = p_i - R(t_i - t'_i)$$

   where $t(i)$ and $t'(i)$ are given by:

   $$\begin{aligned} t(i) &= \frac{L(i)}{v_{max} - (W(i-1) + w_i)\frac{v_{max}-v_{min}}{C}} \\ t'(i) &= \frac{L(i)}{v_{max} - W(i-1)\frac{v_{max}-v_{min}}{C}} \end{aligned}$$

   and:

   $$L(i) = \begin{cases} 0 & \textit{for i = 1} \\ \sum_{n=i}^{N} \delta(n) & \textit{otherwise} \end{cases}$$

Algorithm 3: Pseudo-polynomial solver for KP

```
1: P[0] is a vector of initial profits; if not supplied to the procedure P[0] ← 0
2: for i ← 1 to I do
3:    for w ← 0 to C do
4:       if w_i > w then
5:          P[i][w] ← P[i − 1][w]
6:       else
7:          P[i][w] ← max(P[i − 1][w], P[i − 1][w − w_i] + p_i)
8: m ← 0
9: for w ← 0 to C do
10:   if P[I][w] > P[I][m] then
11:      m ← w
12: w ← m
13: p ← P[C][w]
14: σ ← {}
15: for t ← 0 to I − 1 do
16:   i ← I − t
17:   if p > P[i − 1][w] then
18:      σ ← σ ∪ {i}
19:      w ← w − w_i
20:      p ← p − p_i
21: return σ
```

The items whose relaxed profit is not strictly positive are not taken into consideration when forming an instance of KP.

Instances of KP are solved exactly by the dynamic programming approach (Algorithm 3).

3. Heuristic reduction from KRP to KP and weighted greedy approach to KP.

   The reduction proceeds like in the above. To solve an instance of KP we use a variant of the greedy approach — the items are sorted according to the ratio $\frac{\overline{p}_i}{w_i^\Theta}$, where $\Theta \geq 0$ is a weighting parameter, and then greedily packed into the knapsack (Algorithm 4). We use a constant set of weighting parameters $\Theta \in \{0, \frac{1}{e}, 1, e\}$ and return the picking plan for the best parameter Θ. Observe that for $\Theta = 0$ we get the usual naive algorithm ("best value first"), and for $\Theta = 1$ we get the usual greedy algorithm ("best ratio first"). It may be shown that by using these two parameters only, we get a 1.5-approximation scheme.

We have implemented the following algorithms for TSKP component.

1. Exact solver for TSKP implemented by the usual by the usual branch-and-bound technique.

2. Heuristic reduction from TSKP to TSP and exact solver for TSP.

   Given an instance of TSKP, we create an instance of TSP with the same nodes, but whose distances are substituted by the time needed for the travel:

$$\overline{d}(\pi_n, \pi_{n+1}) = \frac{d(\pi_n, \pi_{n+1})}{v_{max} - W(n)\frac{v_{max} - v_{min}}{C}}$$

Algorithm 4: Heuristic solver for KP

$P^* \leftarrow 0, \sigma^* \leftarrow \{\}$
for $\Theta \in \{0, \frac{1}{e}, 1, e\}$ **do**
  $s \leftarrow$ sorted list of items according to $\frac{p_i}{w_i^\Theta}$
  $W \leftarrow 0, P \leftarrow 0, \sigma \leftarrow \{\}$
  for $i = 1$ **to** I **do**
    if $W + w_{s_i} < C$ **then**
      $W \leftarrow w_{s_i}$
      $P \leftarrow P + p_{s_i}$
      $\sigma \leftarrow \sigma \cup \{s_i\}$
  if $P > P^*$ **then**
    $P^* \leftarrow P$
    $\sigma^* \leftarrow \sigma$
return σ^*

The instances of TSP obtained in this way are exactly solved by the branch-and-bound technique.

3. Heuristic reduction from TSKP to TSP and heuristic TSP.

   The reduction proceeds like in the above. Instances of TSP are solved by the state-of-the-art solver for TSP (that is, Concorde: Cook (1995)).

5.2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (see: Abramson (1991)) is a metaheuristic for decision processes. Originally, it has been proposed to play board games such as Hex (see: Arneson et al. (2010)), Othello (see: Robles et al. (2011)) and most notably: GO (see: Coulom (2009)). It has been also successfully applied to some optimization problems, including a variant of TSP (see: Perez et al. (2012)) and VRP (see: Takes (2010)).

The idea behind Monte-Carlo Tree Search is to sample random solutions and based on their quality make the most promising local decision. Here we apply this idea to solve TTP. Starting from the initial city and the empty knapsack we interchangeably perform the following two steps:

- (TSP Phase) extend the current route by a node n and run a number of random simulations with the extended route; calculate the best profit p_n from all simulations; add to the route node n having maximal profit p_n
- (KP Phase) for every set of items $J \subseteq I_n$ that are available at the current city n, extend the knapsack by J and run a number of random simulations with the extended knapsack; calculate the best profit p_J from all simulations; add to the knapsack items J having maximal profit p_J

until a complete tour is constructed (Algorithm 5).

5.3 Ant Colony Optimization

Methods based on Ant Colony Optimization (ACO) were proposed in early '90 to solve the Traveling Salesman Problem (see: Dorigo and Blum (2005)), and later have been extended to problems like: the Scheduling Problem (see: Martens et al. (2007)), the Assignment Problem (see: Ramalhinho Lourenço and Serra (2002)), the Vehicle Routing

Algorithm 5: Monte-Carlo Tree Search for TTP

```
1: π ← ∅
2: σ ← ∅
3: for n ← 1 to N do
4:    E(π) ← possible extensions of partial cycle π
5:    for π_E ∈ E(π) do
6:       P(π_E) ← −∞
7:       for k ← 1 to maxIter do
8:          π_+ ← extend π followed by π_E to a random cycle
9:          σ_+ ← extend σ with random items
10:         P_+ ← profit(π_+, σ_+)
11:         if P_+ > P(π_E) then
12:            P(π_E) ← π_+
13:            P(π_E) ← P_+
14:   π ← extend partial cycle π with such π_E that maximizes estimated profit P(π_E)
15:   I_{π_E} ← set of items avaliable at city π_E
16:   for i ∈ I_{π_E} do
17:      S ← 0
18:      for k ← 1 to maxIter do
19:         π_+ ← extend π to a random cycle
20:         σ_+ ← extend σ with item i and some random items
21:         σ_† ← extend σ with some random items
22:         P_+ ← profit(π_+, σ_+)
23:         P_† ← profit(π_+, σ_†)
24:         if P_+ > P_† then
25:            S ← S + 1
26:         else
27:            S ← S − 1
28:      if S > 0 then
29:         σ ← σ ∪ {i}
30: return π, σ
```

Problem (see: Toth and Vigo (2002)), the Set Cover Problem (see: Leguizamon and Michalewicz (1999)), and many more.

The general idea behind ACO is to iteratively perform the following steps:

- construct a number of random solutions; the solutions are constructed incrementally by making local choices with some probabilities ρ
- evaluate solutions and adjust probabilities ρ of local choices — increase the probabilities of choices that have led to better solutions

until a stopping condition is satisfied. For example, in an ACO approach to TSP, a tour in a graph G is constructed by locally choosing the next edge from a given node. The edge (i, j) from i to j is chosen with probability:

$$\rho(i,j) = \frac{\tau^{\alpha}(i,j)d(i,j)^{-\beta}}{\sum_{(i,k)\in G}\tau^{\alpha}(i,k)d(i,k)^{-\beta}}$$

where $\alpha, \beta \geq 0$ are parameters of the algorithm and $\tau(i,j)$ indicates "the amount of pheromone" on edge (i,j). The pheromone is updated during each iteration of the algorithm for each random solution π^r according to the following rule:

$$\tau'(i,j) = (1-\gamma)\tau(i,j) + \sum_k H(\pi^r)^{-1}$$

where $0 \leq \gamma \leq 1$ is the pheromone decay parameter, and $H(\pi^r)$ is the total length of the tour π^r, i.e.:

$$H(\pi^r) = \sum_{i=1}^{N} d(\pi_i^r, \pi_{i+1}^r)$$

Alaya et al. (2004), and Fidanova (2002) discuss applications of ACO to the Knapsack Problem. A local choice corresponds to picking an item. The probability of picking an item i is given by:

$$\rho(i) = \frac{\tau^\alpha(i) q(i)^{-\beta}}{\sum_{1 \leq i \leq n} \tau^\alpha(i) q(i)^{-\beta}} \tag{7}$$

where $q(i)$ is a function of "attractiveness" of item i, usually given by:

$$q(i) = \begin{cases} 0 & \textit{if item } i \textit{ cannot be picked anymore} \\ \frac{p_i}{w_i} & \textit{otherwise} \end{cases} \tag{8}$$

and the pheromone $\tau(i)$ is updated during each iteration of the algorithm for each random solution σ^r according to a rule similar to the above:

$$\tau'(i) = (1-\gamma)\tau(i) + \sum_k P(\sigma^r)$$

where $P(K^r)$ is the total profit of items in K^r.

Algorithm 6 uses ideas from Alaya et al. (2004), and Dorigo and Blum (2005) to solve TTP. Because there are non-trivial interactions between TSP and KP components in TTP, we had to apply several modifications:

1. Because the cost of traveling between cities depends on the current weight of the Knapsack, we build first a random solution to the KP part of the problem, and then extend it with a random tour. The probability of picking an item is given as in Formula 7:

   $$\overline{\rho}(i) = \frac{\overline{\tau}(i)^\alpha(i) q(i)^{-\beta}}{\sum_{1 \leq i \leq n} \overline{\tau}(i)^\alpha(i) q(i)^{-\beta}}$$

   but the probability of moving from city i to city j is defined according to the time of the travel instead of its distance:

   $$\rho(i,j) = \frac{\tau^\alpha(i,j) t(i,j)^{-\beta}}{\sum_{(i,k) \in G} \tau^\alpha(i,k) t(i,k)^{-\beta}}$$

2. Contrary to KP, in TTP an optimal solution may consist of less items than it is allowed by the capacity of the knapsack (i.e. because the weight of the knapsack impacts the speed of the thief, dropping an item from a solution may lead to a better solution). Therefore ACO has to discover an upper bound on the total weight of items in the knapsack. If W is the capacity of the knapsack, then we use

Algorithm 6: Ant Colony Optimization for TTP

$P^* \leftarrow -\infty$
$\tau(i,j) \leftarrow \epsilon$
$\overline{\tau}(i) \leftarrow \epsilon$
$\overline{\overline{\tau}}(i=x) \leftarrow \epsilon$
for $k \leftarrow 1$ **to** $MaxIter$ **do**
  for $r \leftarrow 1$ **to** $PopultaionSize$ **do**
    $\sigma \leftarrow \emptyset$
    $\pi \leftarrow \emptyset$
    $\overline{\overline{\rho}} \leftarrow \frac{\overline{\overline{\tau}}(i=x)}{\overline{\overline{\tau}}(i=1)+\overline{\overline{\tau}}(i=0)}$
    $U \leftarrow \min(C, random(\overline{\overline{\rho}}))$
    while true do
      $\overline{\rho(i)} \leftarrow \frac{\overline{\tau}^{\alpha}(i)q(i)^{-\beta}}{\sum_{1\leq i\leq n}\tau^{\alpha}(i)q(i)^{-\beta}}$
      $i \leftarrow$ *sellect a random item with probability* $\overline{\tau}$ *and upper bound* U
      if no item can be selected then break
      $\sigma = \sigma \cup \{i\}$
    for $n \leftarrow 1$ **to** N **do**
      $\rho(i,j) \leftarrow \frac{\tau^{\alpha}(i,j)t(i,j)^{-\beta}}{\sum_{(i,k)\in G}\tau^{\alpha}(i,k)t(i,k)^{-\beta}}$
      $i \leftarrow$ *select at random the next city to travel with probability* τ
      if there is no city to select then break from the loop
      $\pi(n) = i$
    $P(r) \leftarrow$ *profit*(π, σ)
    if $P(r) > P^*$ **then**
      $P^* \leftarrow P(r)$
      $\pi^* \leftarrow \pi$
      $\sigma^* \leftarrow \sigma$
  $\tau(i,j) \leftarrow (1-\gamma)\tau(i,j) + \sum_r P(r)^{-1}$
  $\overline{\tau}(i) \leftarrow (1-\gamma)\overline{\tau}(i) + \sum_r P(r)^{-1}$
  $\overline{\overline{\tau}}(i=x) \leftarrow (1-\gamma)\overline{\overline{\tau}}(i=x) + \sum_r P(r)^{-1}$
return π^*, σ^*

$\lfloor \log(W) \rfloor + 1$ bits to encode the upper bound on the weight of items. The probability that the i-th bit of the upper bound is x is:

$$\overline{\overline{\rho}}(i=x) = \frac{\overline{\overline{\tau}}(i=x)}{\overline{\overline{\tau}}(i=1) + \tau(i=0)}$$

and the pheromone $\overline{\overline{\tau}}(i=x)$ is updated during each iteration of the algorithm for each random solution σ_x^r with the upper bound having i-th bit set to x:

$$\overline{\overline{\tau}}'(i=x) = (1-\gamma)\overline{\overline{\tau}}(i=x) + \sum_{\sigma_x^r} P(\sigma_x^r)$$

3. The initial pheromone is uniformly distributed across cities and items.

5.4 Exact solver for TTP

Exact solver implements the branch-and-bound technique to solve TTP. For each Hamiltonian cycle an instance of the Knapsack Problem is considered, where the value

of an item is amortized by the minimal cost required for its transport. Branch-and-bound is used both in generating Hamiltonian cycles, as well as in solving the Knapsack Problems. Exact solver guarantee optimality of produced solutions and serves as a benchmark algorithm against which other algorithms are compared.

6 Benchmarks

To compare performance of algorithms for TTP, we prepared in Bonyadi et al. (2014) a generic framework for generating classes of TTP-instances. Each class was composed of three independent components: *meta*, *TSP* and *KP*. We recall from Bonyadi et al. (2014) the explanation of these components in the below. Depending on the configuration parameters of the components, one is able to create separate classes of TTP-instances. In addition to Bonyadi et al. (2014), the *TSP* component includes well-known benchmark instances from a public database.

1. (Meta) This component describes graph and items independent parameters of the Traveling Thief Problem: a natural number C describing the capacity of the knapsack, a non-negative real number R indicating the rent ratio, and two positive real numbers $v_{min} \leq v_{max}$ corresponding to the minimal and maximal speed of the traveler. Thanks to these parameters we can adjust the coupling between the subcomponents of TTP (if $v_{max} = v_{min}$ the subproblems are "fully sequential" — there are no interaction between subproblems) and their relative importance (if the rent rate $R = 0$, the solution to the TSP part may be completely ignored, as it has no impact on the objective function of TTP)

2. (TSP) This component describes the graph of the Traveling Thief Problem. Such a graph is a pair $\langle V, E\rangle$ consisting of a set of nodes $m, n \in V$ (called cities), and a set of edges $m \overset{d(m,n)}{\longrightarrow} n \in E$ from m to n (called distances). We have used four sources of graphs (for more details see: Bonyadi et al. (2014)): *random graphs* Solomonoff and Rapoport (1951) (distances are independently assigned according to some priori distribution), *Euclidean graphs* (the nodes of the underlying graph are embedded in some low-dimensional Euclidean space), *Hamiltonian-dense graphs* (the number of paths that cen be extended to the full Hamilttonian cycle is relatively high; the main motivation behind this class of graphs is to make the problem of finding Hamiltonian-cycles easy), a class based on a well-known set of benchmark for TSP:

   `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`

3. (KP) This component describes the set of items together with the availability relation of the Traveling Thief Problem. An item i is a pair of natural numbers $\langle w_i, p_i\rangle$, where w_i is called the weight of the item, and p_i is called the profit of the item. The availability relation between the cities V and a set $\{1, 2, \ldots, I\}$ says that the i-th item is available at city $n \in V$ iff $a(n, i)$ is true. We have used two classes of KP instances: *uncorrelated weights and values* (weights, values and availability of the items are independently sampled from some priori distributions) and *greedy-prof* (because a KP-instance in which values and weights of items are uncorrelated can be easily solved by the greedy algorithm to high quality solution, see: Pisinger (2005), we generated KP-component instances that are resistant to such approaches).

The parameters of the benchmark instances are set to the same values as in Bonyadi et al. (2014).

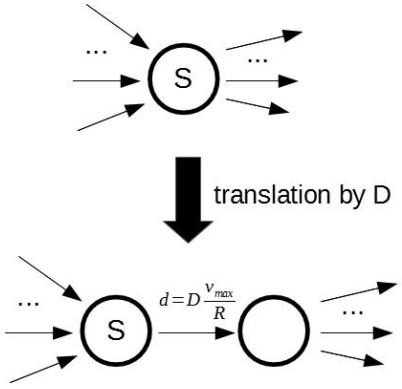

Figure 2: Translation of profits by a non-negative constant D. The starting node S is split into two nodes S' and S'' with the distance from S' to S'' set to $D\frac{v_{max}}{R}$, and such that: S' inherits input edges from S, whereas S'' inherits output edges from S.

7 Experimental Results

In order to compare algorithms for TTP, we generated various instances with different number of cities (from 3 to 76) and items (from 10 to 146) with the parameters listed in Section 6. The full set of benchmark instances together with the scripts to generate them is available at the website (see: Przybylek (2015)).

Before we present the results, we have to face one additional issue — the instances of TTP are not normalised, and in fact, the structure of the TTP itself makes it impossible to perform normalisation of its instances. Therefore, there is no direct way to compare relative performances of different algorithms on different instances. Let us recall the objective function of TTP (Equation 1):

$$P = \sum_{i \in dom(\sigma)} p_i - R \sum_{n=1}^{|V|} t(\pi_n, \pi_{n+1})$$

Observe that for any positive constant K, one may rescale the values p_i and R in an instance by K, obtaining another instance, whose solutions are exactly the same, but profits of the solutions are rescaled by K (thus, the qualities of the solutions are preserved by the rescaling operation). Moreover, the instances are not *localised*, which means that for any non-negative constant D and any instance of TTP, one may build an equivalent instance whose solutions have profits translated by D — it suffices to substitute the starting node by a pair of nodes with a single edge of an appropriate distance (Figure 2).

To overcome this obstacle, we first perform a localisation of solutions at estimated average solutions, and then use for the measure profits relative to the optimal profits. In more detail, let us assume that for a given instance the profit of an optimal solution is P^*, the average profit is $P^\#$ and our algorithm produced a result with profit P. Then the quality q of the solution returned by the algorithm is computed in the following way:

$$Q = \frac{P - P^\#}{P^* - P^\#}$$

Benchmark	Exact	Average	CoSolver		CoSolver E		CoSolver H		MCTS		ACO	
	Profit	*Profit*	*Profit*	*Q*	*Profit*	*Q*	*Profit*	*Q*	*Profit*	*Q*	*Profit*	*Q*
Euclidean	-230563	-317652	-241667	87%	-241667	87%	-230585	100%	-248449	79%	-244484	84%
	-18210	-30219	-18210	100%	-18210	100%	-19918	86%	-21683	71%	-22087	68%
	-38782	-48162	-57438	-99%	-66482	-195%	-38833	99%	-39560	92%	-39136	96%
	-155161	-205688	-155161	100%	-155815	99%	-155334	100%	-155334	100%	-156253	98%
	5038	323	5009	99%	5009	99%	4981	99%	2838	53%	2688	50%
	-36042	-51009	-36042	100%	-36042	100%	-38696	82%	-39101	80%	-40688	69%
	1289	-854	1289	100%	788	77%	538	65%	486	63%	228	50%
	-122329	-202694	-143547	74%	-159672	54%	-122329	100%	-134387	85%	-151206	64%
Euclidean avg	-74345	-106994	-80721	70%	-84012	53%	-75022	91%	-79399	78%	-81367	72%
Dense	-88984	-116443	-93952	82%	-93952	82%	-89299	99%	-96257	74%	-98218	66%
	-32662	-82552	-32921	99%	-32860	100%	-59606	46%	-63856	37%	-59136	47%
	-25346	-69790	-36360	75%	-36360	75%	-49236	46%	-53125	37%	-48176	49%
Dense avg	-48997	-89595	-54411	86%	-54391	86%	-66047	64%	-71079	49%	-68510	54%
Small	17274	-19338	17024	99%	17024	99%	11590	84%	-4283	41%	-1482	49%
	-38181	-63763	-38181	100%	-38659	98%	-38181	100%	-44329	76%	-44446	76%
	-17695	-24310	-19277	76%	-19277	76%	-17771	99%	-18549	87%	-18561	87%
	-30616	-38015	-34807	43%	-35259	37%	-30796	98%	-31832	84%	-30796	98%
	-63706	-72850	-75158	-25%	-75291	-27%	-63706	100%	-65071	85%	-63706	100%
	-58489	-77757	-63518	74%	-63641	73%	-59258	96%	-61949	82%	-63230	75%
	-32946	-74510	-34105	97%	-34817	95%	-32946	100%	-42821	76%	-50334	58%
Small avg	-32051	-52935	-35432	66%	-35703	65%	-33010	97%	-38405	76%	-38936	77%
Random	-19428	-34468	-22100	82%	-21945	83%	-19591	99%	-23082	76%	-24401	67%
	-20176	-38693	-28872	53%	-28872	53%	-20482	98%	-22229	89%	-23556	82%
	7369	1696	7351	100%	7351	100%	7108	95%	3731	36%	4222	45%
	5521	1632	5521	100%	5521	100%	5507	100%	2603	25%	3043	36%
	2104	470	2104	100%	2104	100%	2085	99%	670	12%	740	17%
	9969	1661	9955	100%	9964	100%	8221	79%	2741	13%	3105	17%
	8834	1023	8834	100%	8830	100%	8833	100%	3406	30%	3331	30%
Random avg	-829	-9525	-2458	91%	-2435	91%	-1189	96%	-4594	40%	-4788	42%
KP Centric	39937	2459	39935	100%	39937	100%	29281	72%	28917	71%	17926	41%
	69336	1532	69335	100%	69331	100%	36542	52%	38272	54%	17117	23%
	90025	1828	89992	100%	90019	100%	81479	90%	55366	61%	18346	19%
	69484	1906	69478	100%	69484	100%	62682	90%	40603	57%	17778	23%
	74234	1824	74233	100%	74234	100%	60684	81%	46817	62%	21952	28%
	65531	1850	65524	100%	65524	100%	65043	99%	34558	51%	17983	25%
	80049	2629	80046	100%	80046	100%	56291	69%	52480	64%	19881	22%
	59604	1611	59597	100%	59604	100%	51167	85%	30234	49%	17478	27%
	50766	2498	50743	100%	50757	100%	27678	52%	36377	70%	18115	32%
	27358	886	27358	100%	27358	100%	17082	61%	21651	78%	11272	39%
KP cent. avg	62632	1902	62624	100%	62629	100%	48793	75%	38527	62%	17785	28%
Greedy	1178064	49008	1178064	100%	1178064	100%	1002796	84%	1168390	99%	1008409	85%
	1551296	61677	1551296	100%	1551296	100%	1342664	86%	1447922	93%	1353361	87%
	659186	29237	659186	100%	659186	100%	576326	87%	650737	99%	580318	87%
	1384486	79465	1384486	100%	1384486	100%	1186825	85%	1331286	96%	1195859	86%
	1401970	58205	1401970	100%	1401970	100%	1224622	87%	1340589	95%	1233473	87%
	1188129	54522	1188129	100%	1188129	100%	1020886	85%	1131986	95%	1038696	87%
	1143685	40462	1143685	100%	1143685	100%	997422	87%	1131531	99%	1006294	88%
	2099247	106191	2099247	100%	2099247	100%	1812908	86%	2056908	98%	1843659	87%
	771036	36070	771036	100%	771036	100%	649061	83%	721852	93%	662999	85%
	1425409	87466	1425409	100%	1425409	100%	1225759	85%	1367081	96%	1241604	86%
Greedy avg	1264122	57204	1264122	100%	1264122	100%	1090390	86%	1220133	96%	1102563	87%
Average	276820	-20903	274545	87%	273924	84%	233100	86%	257425	70%	226222	61%

Table 1: Performance of the algorithms on various benchmarks. Columns: Exact shows optimal solutions, Average shows an average solution from 1000 random solutions, CoSolver, CoSolver E, CoSolver H show solutions obtained by various CoSolver algorithms, MCTS shows solutions obtained by Algorithm 5, and ACO shows solutions obtained by Algorithm 6.

Any reasonable algorithm should return a solution having quality between 0% and 100%. Here 0% means that the algorithm produced an average (i.e. random) solution, and 100% means that the algorithm produced an optimal solution. One may actually think of this quality as of the "smartness" of an algorithm, where 0% does not require any work (i.e. statistically, it suffices to construct a random solution), negative values indicate that the algorithm has been misled (i.e. it has produced solutions worse than the solutions that do not require any computation) and values $0 - 100\%$ measure the real effectiveness of the algorithm. Nonetheless, there is one problem here — the average profit $P^{\#}$ depends, of course, on the probability distribution on the spaces of possible solutions; and since a solution to TTP comprises of a solution to the Hamiltonian Problem, which is NP-complete, one should not expect that a random solution to TTP generated according to any polynomial distribution would be feasible. We were forced to use a different strategy — first we generated a random Hamiltonian cycle, and then supplied it with randomly chosen items. Therefore, one has to remember, that a "random solution" is not-that-easy to obtain — there is a highly non-trivial problem underlying the random samples.

For large competitive instances, where we could not obtain the exact solutions in any reasonable time, we present the "gain" obtained by an algorithm over an average random solution:

$$Gain = P - P^{\#}$$

Our algorithms are applied to the benchmark problems and their results are compared. The methods that use any kind of non-determinism (Ant Colony Optimization, Monte-Carlo Tree Search) were run sixteen times and the average solutions have been taken for the final results. In addition, for the main set of benchmark problems, where we could obtain exact results, we present a graph of the performance of non-deterministic algorithms with error bars indicating the best and worst solutions and the 95% confidence interval for every instance (see Figures 3 and 4). Figure 4 shows that the confidence intervals are usually quite narrow, allowing for a clear comparison of performance of the various methods, which justifies our choice of the number of runs for the non-deterministic heuristics.

We also designed an exhaustive search algorithm that solves the main benchmark set to the optimality, and estimated an average solution of each of the benchmark problems. The benchmarks are divided on three classes. The full set of results is available at the website (see: Przybylek (2015)).

7.1 Typical

This class of benchmarks contains typical instances of TTP as described in the previous section. The results are presented in Table 1, where Average is an estimated average profit, CoSolver is the original CoSolver algorithm as introduced in Bonyadi et al. (2014), CoSolver Exact is a variant of CoSolver based on a heuristic method for TSKP component and exact solver for KRP component (Algorithm 2), CoSolver Heuristic is a variant of CoSolver based on a heuristic methods for both of its components, MCTS is a heuristic based on Monte-Carlo Tree Search (Algorithm 5), and ACO is a method based on Ant Colony Optimization (Algorithm 6). The table shows that CoSolver Heuristic and MCTS never produce bad solutions — the worst are almost twice as good as the average solution. Moreover, MCTS outperforms ACO in most cases.

The two last set of benchmarks was tuned to mislead "greedy heuristics" of the KP-subcomponent. Notice that the solutions generated by algorithms that are sensitive to greedy-proof instances — i.e. CoSolver Heuristic and Monte Carlo Tree Search —

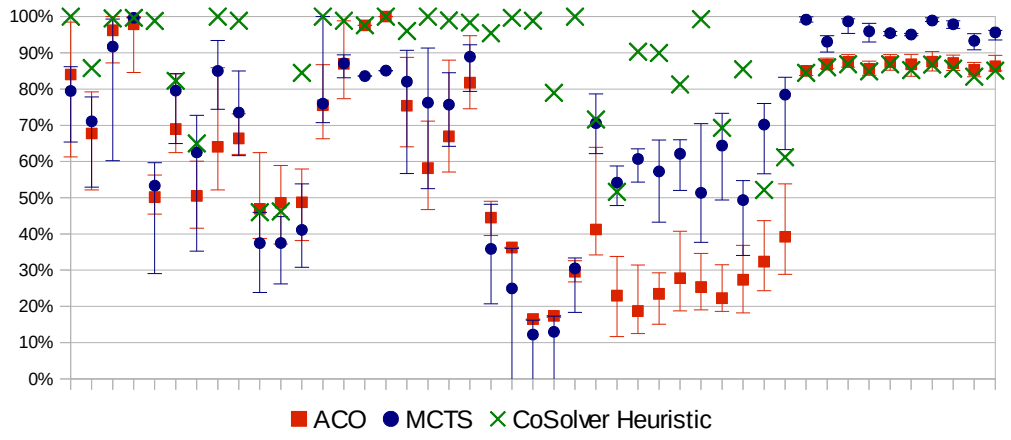

Figure 3: Performance of CoSolver Heuristic against ACO and MCTS with error bars on instances from Table 1.

are still of a reasonable quality.

Figure 3 shows performances of CoSolver Heuristic against ACO and MCTS with error bars indicating the best and worst solutions for every instance, and Figure 4 shows the 95% confidence intervals.

It is worth noticing that for many classes of problems the version of CoSolver that is based on purely heuristic components (CoSolver Heuristic) performs better than the original CoSolver on the average. Moreover, Table 1 shows that both the original CoSolver and CoSolver Exact may get misled and produce a worse than random solutions (red cells in Table 1). One may explain this phenomenon by the fact that although heuristic components give partial solutions that are locally worse than optimal, the solutions are also less sensitive to further changes and, therefore, can potentially lead to better global solutions. Additionally, CoSolver Heuristic greatly improves the scalability of CoSolver without compromising quality. Observe, also, that while not as good as methods based on pre-existing components, Monte-Carlo Tree Search may provide an interesting alternative in case when there is a little knowledge about sub-components of the initial problem, or the coupling between the sub-components is high making negotiation protocols infeasible.

7.2 Known TSP Benchmarks

To build a competitive set of benchmarks for TTP, we decided to use a well-known public database of symmetric and asymmetric TSP instances:

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`

and extend them with randomly generated items. The instances, however, are too big

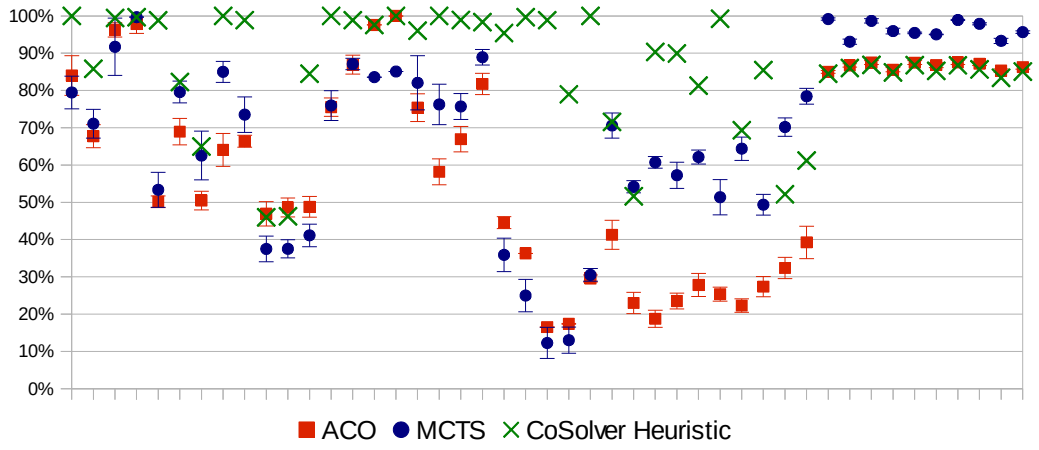

Figure 4: Performance of CoSolver Heuristic against ACO and MCTS with 95% confidence intervals on instances from Table 1.

Benchmark	Average	CoSolver H		MCTS		ACO	
	Profit	*Profit*	*Gain*	*Profit*	*Gain*	*Profit*	*Gain*
att48	-1189	-165	1024	-682	507	-929	260
bayg29	-199	-67	132	-113	87	-155	44
bays29	-200	-67	132	-119	80	-159	41
berlin52	-585	-57	528	-366	219	-482	103
br17	-289	19	308	19	308	-98	191
brazil58	-1205	-160	1045	-721	484	-967	237
burma14	-79	-40	40	-45	34	-56	23
dantzig42	-874	-126	748	-443	431	-695	179
eil51	-503	4	507	-217	286	-381	122
eil76	-2039	-208	1831	-1306	734	-1688	351
fri26	-237	-82	155	-112	125	-183	54
ft53	-615	-162	453	-434	181	-531	84
ft70	-973	-383	590	-625	347	-841	132
ftv33	-239	-70	169	-125	114	-191	48
ftv35	-721	-214	506	-453	267	-588	133
ftv38	-122	-36	86	-64	59	-94	28
ftv44	-986	-166	820	-630	357	-800	186
ftv47	-975	-167	807	-628	346	-800	175
ftv55	-1671	-203	1468	-1070	600	-1403	268
ftv64	-1128	-223	904	-764	364	-958	170
ftv70	-2304	-404	1900	-1523	782	-1929	375
gr17	-100	40	140	18	118	-11	89
gr21	-173	-61	112	-94	79	-132	41
gr24	-172	-63	109	-52	120	-112	60
gr48	-1053	-221	833	-620	434	-846	208
hk48	-1081	-171	910	-673	408	-865	215
p43	-955	-168	787	-190	765	-522	432
pr76	-2350	-244	2106	-1483	868	-1927	423
ry48p	-960	-166	794	-577	382	-747	212
swiss42	-748	-117	631	-440	308	-588	160
ulysses16	-93	-52	42	-18	76	-42	51
ulysses22	-220	-93	127	-125	94	-168	52
Average	-782	-134	648	-459	324	-622	161

Table 2: Performance of the algorithms on known graphs. Columns: Average shows an average solution from 1000 random solutions, CoSolver H shows solutions obtained by CoSolver Heuristic, MCTS shows solutions obtained by Algorithm 5, and ACO shows solutions obtained by Algorithm 6.

Benchmark	Average	CoSolver H		MCTS		ACO	
	Profit	*Profit*	*Gain*	*Profit*	*Gain*	*Profit*	*Gain*
Highly dependant	120211	335000	214789	389007	268796	314353	194142
Dependant	121727	425000	303273	401650	279923	336261	214534
Balanced	121482	455000	333518	405873	284391	330917	209435
Moderately sequential	130486	465000	334514	418095	287609	349277	218791
Fully sequential	137186	470000	332814	420003	282817	355598	218411

Table 3: Performance of the algorithms wrt sequentiality. Columns: Average shows an average solution from 1000 random solutions, CoSolver H shows solutions obtained by CoSolver Heuristic, MCTS shows solutions obtained by Algorithm 5, and ACO shows solutions obtained by Algorithm 6.

to be solved to optimality by the exact solver, or even to be solved by the CoSolvers with exact components. Therefore, we produced the results for CoSolver Heuristic, MCTS and ACO only. The benchmarks are presented in Table 2. Columns *Profit* describe the profit obtained by a given algorithm, and columns *Gain* describe the "gain" obtained by an algorithm with respect to the average solution.

This table confirms that CoSolver Heuristic outperforms MCTS and ACO, and that MCTS is fairly better than ACO.

7.3 Coupling based

We have also tested performance of our algorithms with respect to coupling between sub-components. An instance of TTP is "sequential" if a good solution can be obtained by independently solving its first component, and on top of it solving its second component. We have prepared six sets of instances with increasing level of sequentiality and applied both the CoSolver and MCTS algorithms to them. The normalised results are shown in Table 3.

The impact of "coupling" of TTP components on the difficulty of obtaining good results using our heuristics is clear. TTP instances in which components are more dependent on each other are more difficult to solve well. This result gives insight into the difficulty of other multi-hard problems. Even though multi-hard problems may be in the same complexity class as their components, they can be more difficult than each of the components and this difficulty increases with increasing component interdependence.

Also, for most of the considered instance types, CoSolver H (the algorithm that aims to exploit multi-hard problem structure) does a better job than MCTS and ACO. However, results also suggest that MCTS-based algorithms perform better on problems that have large cohesion between their subcomponents (highly dependent). This shows that further work is needed to design algorithms that can better exploit problem structure. Recall that CoSolver's design relies on good methods to "negotiate" a solution between solvers for the components of a multi-hard problem. A very high coupling (dependency) between the components of a multi-hard problem seems to make this "negotiation" less effective.

8 Conclusions and Further Work

In this paper we have introduced the concept of multi-hardness — i.e. problem that are non-trivial combinations of classical hard problems. We have studied algorithms that exploit the structure of multi-hard problems through an evaluation of such algorithms for solving TTP, a model multi-hard problem. We have extended the idea of CoSolver

by incorporating heuristics instead of exact solvers for the Traveling Salesman Problem and Knapsack Problem components of the Traveling Thief Problem. Moreover, we have introduced a new promising heuristic for multi-hard problems that bases on Monte-Carlo Tree Search. We also examined a heuristic based on Ant Colony Optimisation. We have developed a set of publicly available benchmarks for TTP and have used it to compare the heuristics against each other.

Our experiments show that, when it comes to partial solutions, heuristic components may lead to better global solutions, because the results produced by such components are generally "more stable" — i.e. are less sensitive to further changes. In the experiments, the version of CoSolver that is based on purely heuristic components (CoSolver Heuristic) performs better than the original CoSolver on the average. Moreover, CoSolver Heuristic and MCTS never produce bad solutions. We also note that Monte Carlo Tree Search may provide an interesting alternative to CoSolver-based heuristics in case there is a little knowledge about sub-components of the initial problem, or if the coupling between the sub-components is high enough to make any negotiation protocols between sub-components ineffective. Our results confirms the effectiveness of using a decomposition-negotiation approach to multi-hard problems.

The coupling between TTP components has a great impact on the difficulty of obtaining good results. TTP instances in which components are more dependent on each other are more difficult to solve well by our decomposition-based algorithms. This gives insight into the difficulty of other multi-hard problems. Even though multi-hard problems may be in the same complexity class as their components, they can be more difficult than each of the components and this difficulty increases with increasing component interdependence. Also, for most of the considered instance types, CoSolver Heuristic (the algorithm that aims to exploit multi-hard problem structure) does a better job than MCTS and ACO. However, results also suggest that MCT-based algorithms perform better on problems that have large cohesion between their subcomponents (i.e. are highly dependent). This shows that further work is needed to design algorithms that can better exploit problem structure. Recall that CoSolver's design relies on good methods to "negotiate" a solution between solvers for the components of a multi-hard problem. A very high coupling (dependency) between the components of a multi-hard problem seems to make this "negotiation" less effective. We believe that better methods for such "negotiation" may still be discovered.

Our long-term goal is to provide a broad new methodology for integration of multi-hard problems progressing from simpler couplings of silos and sequences, to heterogeneous highly connected models. In the future work we will be interested in extending our model Traveling Thief Problem with additional subcomponents and various aspects that may be found in real-world systems (such as incompleteness and uncertainty of information, or information that changes over time) and developing new decomposition-based methodologies for such extensions. We will be also interested in validating our methods in an industrial environment.

References

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

Alaya, I., Solnon, C., and Ghédira, K. (2004). Ant algorithm for the multi-dimensional knapsack problem. In *International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*. Citeseer.

Arneson, B., Hayward, R., and Henderson, P. (2010). Mohex wins hex tournament. *Icga Journal*, 33(3):181.

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

Bonyadi, M. R. and Michalewicz, Z. (2016). Evolutionary computation for real-world problems. In *Challenges in Computational Statistics and Data Mining*, pages 1–24. Springer.

Bonyadi, M. R., Michalewicz, Z., and Barone, L. (2013). 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.

Bonyadi, M. R., Michalewicz, Z., Neumann, F., and Wagner, M. (2016). Evolutionary computation for multicomponent problems: opportunities and future directions. *CoRR*, abs/1606.06818.

Bonyadi, M. R., Michalewicz, Z., Przybylek, M. R., and Wierzbicki, A. (2014). 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. ACM.

Borovik, A. V. (2009). *Mathematics under the Microscope: Notes on Cognitive Aspects of Mathematical Practice*. American Mathematical Society.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2015). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*.

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

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

Cook, W. J. (1995). A computer code for tsp. `http://www.math.uwaterloo.ca/tsp/concorde.html`.

Coulom, R. (2009). The monte-carlo revolution in go. In *The Japanese-French Frontiers of Science Symposium (JFFoS 2008), Roscoff, France*.

Davis, L. (1985). 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.

Derrouiche, R., Neubert, G., and Bouras, A. (2008). Supply chain management: a framework to characterize the collaborative strategies. *International journal of computer integrated manufacturing*, 21(4):426–439.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2):243–278.

Faires, J. and Burden, R. (2003). *Numerical Methods*. Number t. 1 in Numerical Methods. Thomson/Brooks/Cole.

Fidanova, S. (2002). Aco algorithm for mkp using various heuristic information. In *International Conference on Numerical Methods and Applications*, pages 438–444. Springer.

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

Gunawardena, J. (1998). The unreasonable ineffectiveness of mathematics in computer engineering. Research seminar at University of Sydney.

Kearns, M. (2012). Experiments in social computation. *Communications of the ACM*, 55(10):56–67.

Leguizamon, G. and Michalewicz, Z. (1999). A new version of ant system for subset problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2. IEEE.

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

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

Malandraki, C. and Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science*, 26(3):185–200.

Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., and Baesens, B. (2007). Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11(5):651–665.

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

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

Perez, D., Rohlfshagen, P., and Lucas, S. M. (2012). Monte-carlo tree search for the physical travelling salesman problem. In *European Conference on the Applications of Evolutionary Computation*, pages 255–264. Springer.

Pisinger, D. (2005). Where are the hard knapsack problems? *Comp. & Op. Res.*, 32(9):2271–2284.

Poli, R. (1999). Poli seminar abstract. research seminar at Category Theory Research Center, McGill University.

Przybylek, M. R. (2015). Multihard problems. `https://sites.google.com/site/travellingthief`.

Przybylek, M. R., Wierzbicki, A., and Michalewicz, Z. (2016). Multi-hard problems in uncertain environment. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 381–388, New York, NY, USA. ACM.

Ralphs, T. K., Kopman, L., Pulleyblank, W. R., and Trotter, L. E. (2003). On the capacitated vehicle routing problem. *Mathematical programming*, 94(2-3):343–359.

Ramalhinho Lourenço, H. and Serra, D. (2002). Adaptive search heuristics for the generalized assignment problem. *Mathware & soft computing. 2002 Vol. 9 Núm. 2 [-3].*

Robles, D., Rohlfshagen, P., and Lucas, S. M. (2011). Learning non-random moves for playing othello: Improving monte carlo tree search. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 305–312. IEEE.

Solomonoff, R. and Rapoport, A. (1951). Connectivity of random nets. *The bulletin of mathematical biophysics*, 13(2):107–117.

Takes, F. W. (2010). Applying monte carlo techniques to the capacitated vehicle routing problem. In *Proceedings of 22th Benelux Conference on Artificial Intelligence (BNAIC 2010)*.

Talbi, E.-G. (2013). *Metaheuristics for Bi-level Optimization*. Springer Publishing Company, Incorporated.

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

Toth, P. and Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512.

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

Velupillai, K. V. (2005). The unreasonable ineffectiveness of mathematics in economics. *Cambridge Journal of Economics*, 29(6):849–872.

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