J Glob Optim (2017) 68:601–622
DOI 10.1007/s10898-016-0486-5

The reduction of computation times of upper and lower tolerances for selected combinatorial optimization problems

Marcel Turkensteen[1] · **Dmitry Malyshev**[2] · **Boris Goldengorin**[3,4] · **Panos M. Pardalos**[4]

Received: 11 December 2015 / Accepted: 2 December 2016 / Published online: 10 December 2016

Abstract The tolerance of an element of a combinatorial optimization problem with respect to its optimal solution is the maximum change of the cost of the element while preserving the optimality of the given optimal solution and keeping all other input data unchanged. Tolerances play an important role in the design of exact and approximation algorithms, but the computation of tolerances requires additional computational time. In this paper, we concentrate on combinatorial optimization problems for which the computation of all tolerances and an optimal solution have almost the same computational complexity as of finding an optimal solution only. We summarize efficient computational methods for computing tolerances for these problems and determine their time complexity experimentally.

Keywords Discrete optimization · Tolerances · Complexity · Efficient algorithm

1 Introduction

In combinatorial minimization problems (CMPs), we wish to determine a prespecified combination of elements with the minimum costs. This paper considers the usage of so-called

✉ Marcel Turkensteen
matu@econ.au.dk

Dmitry Malyshev
dmalishev@hse.ru

Boris Goldengorin
goldengo@ohio.edu; goldengorin@ufl.edu

Panos M. Pardalos
pardalos@ufl.edu

1 Aarhus University, Fuglesangs Alle 4, Aarhus V, Denmark

2 National Research University Higher School of Economics, 25/12 Bolshaya Pecherskaya Ulitsa, Nizhny Novgorod, Russia 603155

3 Ohio University, Athens, OH 45701, USA

4 University of Florida, 401 Weil Hall, P.O. Box 116595, Gainesville, FL 326116595, USA

tolerances for solving $\mathcal{NP}$-hard problems and addresses computational issues related to tolerances. We focus on the frequently encountered type of CMPs with additive objective functions, where the overall cost of any solution is a sum function of the costs of the selected elements. Many CMPs satisfy the property that the objective function is additive, including many problems in scheduling (cost or time minimization), routing (minimize distance or length), location (minimize total costs, number of locations, total distance, maximize coverage), and network design (minimize total distances).

Such $\mathcal{NP}$-hard CMPs can be solved in different ways; see [49]. Exact methods include mixed integer programming (MIP) type approaches that use a fast generic integer linear programming (ILP) solver such as CPLEX to find optimal solutions. This is often effective, but some problems are better solved using solution methods that do not use an ILP solver, but are problem-specific and exploit properties of a given problem. We call them *combinatorial approaches*, such as dynamic programming and versions of branch and bound. Approximate solution approaches, the (meta-)heuristics, are often combinatorial as well.

Many combinatorial approaches for $\mathcal{NP}$-hard problems use a so-called *relaxation*. We use the definition of the relaxation as a problem that is obtained by deleting (relaxing) a set of constraints that make the original problem difficult to solve; see e.g. [29], p. 517. This includes LP relaxation. Other forms of relaxations of a minimization problem can be obtained by replacing the original objective function with an underestimating function instead of or in addition to replacing the feasible region by one that strictly contains it. An example of this is *Lagrangian relaxation*, where some constraints that make the problem hard to solve are penalized in the objective function [16]. These forms of relaxation, however, are not considered further in this paper.

In the selected form of relaxations, obtained by deleting constraints, the objective function of the relaxation is the same as that of the original CMP, i.e., an additive objective function. If an optimal solution to such a relaxation is feasible for the original problem, it is also optimal to the original problem. If, however, such a relaxation solution is infeasible for the original $\mathcal{NP}$-hard problem, it holds that one or more elements from the relaxation solution should be removed, and/or one or more elements from outside of it have to be included.

The decision in many heuristics and exact methods is on the selection of elements to remove from or include into a solution. A readily available measure is a cost of an element: if we decide to remove a high cost element or include a low cost element, one may expect that the resulting solution has low overall costs. However, the cost is a very *local* measure: if one deletes a high cost element from a solution, it may leave together with low cost elements and it may be replaced by high cost elements in the newly obtained solution.

As an alternative, tolerances can be used in combinatorial approaches, both in heuristics and exact methods. When the cost of a single element changes by sufficiently much, the combination (also called a feasible solution) remains no longer optimal for the new changed problem instance. The maximum change of the cost of a single element such that the current solution remains optimal is called a *tolerance* and its non-negative value is referred to as a *tolerance value*. More precisely, the maximum increase and decrease of the cost of an element are called its *upper tolerance* and *lower tolerance*, respectively; see Sect. 2.

As shown in Sect. 2, it holds for problems with additive objective functions that a tolerance value measures the overall cost change from all the changes in the solution resulting from including or excluding elements. The tolerance value then measures the overall (global) change in solution value resulting from including or excluding an element from our current optimal relaxation solution. That means, for example, that we can determine for which elements their inclusion or exclusion leads to solutions with poor objective values, namely, the elements with large upper and lower tolerance values. A solution method can, based on

tolerance values, avoid putting such elements in its solutions. In that sense, the tolerance value not only contains the cost of a single element, but is a *more global* measure of the consequences of removing or including elements.

We characterize a tolerance-based algorithm as an algorithm that uses tolerance values of a relaxation problem to determine an optimal or approximate solution to an $\mathcal{NP}$-hard problem. For additive CMPs, a tolerance value corresponds to the increase in solution value when an element is forced into a solution or forbidden to enter it. Many tolerance-based algorithms have been introduced [2,5,6,18,45,51] and some of these outperform the state of the art for their respective problems; see Sect. 3.

It is often desirable to have the tolerance values of many elements: the upper tolerance values of those inside the relaxation solution and the lower tolerance values of those outside of it. A first example of this is the finding in the study by [51], which shows that the number of subproblems in their branch and bound algorithm can be reduced by taking the upper tolerances of all elements in the assignment problem (AP; see Sect. 4) solution into account. The study by [17] finds further improvements if lower tolerances are used in addition to upper tolerances. A second example is that the modified Lin–Kernighan heuristic by Helsgaun [28] uses all lower tolerances of the so-called 1-tree problem to determine which elements should be considered for inclusion in candidate tours to the Symmetric Traveling Salesman Problem (STSP). A further improvement, suggested in [45], is to use all upper tolerance values as well.

The disadvantage of using tolerances is that their values have to be computed. If the relaxation problem at hand has an additive cost function, a tolerance value can be obtained by solving a modified instance of the relaxation at hand; see Sect. 2. Therefore, if a given number m tolerances should be determined, m additional problems have to be solved. If this cannot be done efficiently, the computational burden of tolerances can become so large that benefits in more effective search may not lead to solution time reductions compared to methods that use an element's cost value (which is usually part of an instance's input data).

To give an example of the computational complexity of the computation of tolerances, we consider the minimum spanning tree problem (MSTP; see Sect. 5). We relate the complexity to n, which is the number of nodes in the underlying graph of a problem. An MSTP instance on a complete graph can be solved in $\mathcal{O}(n^2)$ time. Since there are $O(n^2)$ elements outside of the optimal MSTP solution, computing all lower tolerance values of all these elements takes $\mathcal{O}(n^4)$ time. Another example is the AP, which can be solved in $\mathcal{O}(n^3)$ time. Computing all lower tolerance values can be done in $\mathcal{O}(n^5)$ time if all the new instances are solved from scratch. Since one can use the current AP solution to determine the solution to the modified instance needed to compute tolerances in $\mathcal{O}(n^2)$ time, this reduces the complexity to $\mathcal{O}(n^4)$. This complexity, however, still puts a heavy burden on the computation of the lower tolerances: if lower tolerances are computed at each iteration in an algorithm, the usage of lower tolerances should reduce the number of iterations to obtain a solution by (at least) a factor $O(n)$ in order to reduce overall computation times.

This challenge is not unique to the MSTP. The paper [17] can improve bounds on solutions to the Asymmetric Traveling Salesman Problem (ATSP) by including lower tolerances, but only through the use of the efficient computation of the lower tolerances from [53], discussed in Sect. 4. The fact that tolerances come at a computational price may form a main barrier to their usage in solution methods. Clearly, if we wish to benefit from the information contained in large sets of upper and lower tolerance values in efficient algorithms, we should be able to compute tolerance values efficiently.

Firstly, we consider the frequently used relaxations of $\mathcal{NP}$-hard problems, namely the AP and the MSTP; see also Sect. 3. For these problems, effective computational methods

for determining all tolerances such as those from [48] and [53] have been developed mostly for the purpose of sensitivity analysis, that is, to determine the changes in cost coefficient values under which the current solution remains optimal. Worst-case complexity results are generally reported for such approaches as the typical question is whether tolerances can be computed in the same time complexity as solving the problem. However, in tolerance-based algorithms, the times for computing tolerance values can form a large share of the running time of an algorithm, in particular when such computations are carried out repeatedly. Actual computation times are relevant, and these have not been determined experimentally, to the best of our knowledge. Further issues are that some methods for computing tolerances are difficult to implement and that some of these methods have received very little attention recently. We address this by reviewing and showing such methods.

Secondly, there are $\mathcal{NP}$-hard problems that are best solved with relaxation problems for which no traditional efficient methods for computing tolerances exist. We present an example of a novel approach for computing all tolerances. The method from [38] uses a specially designed fast tolerances computation approach for the so-called weighted independent set problem (WISP) based on a special polynomially solvable version of this problem as a relaxation, the WISP for trees (WIST). A key ingredient is the fast WIST tolerances computation method introduced in [24]. Another example is the Relaxed Assignment Problem (RAP), briefly discussed in Sect. 7, of which the tolerances are used for the ATSP in, among others, the R-R-GREEDY heuristic introduced in [19].

This paper firstly summarizes methods for computing tolerances, discusses complexity results, and explains in detail how they work. We then determine the reduction in computation times that can be achieved by using smart computation of tolerances in computational experiments. The purpose is to improve the search in combinatorial algorithms; by ensuring that relevant information for the search is available quickly, we aim to improve the choice for the search direction in such algorithms.

The organization of the paper is as follows. Section 2 defines CMPs and their upper and lower tolerances formally and Sect. 3 summarizes tolerance-based algorithms. We present methods for an efficient computation of all upper and lower tolerance values for the AP in Sect. 4, for the MSTP in Sect. 5, and for the WIST in Sect. 6. We briefly discuss the computation of tolerances for other optimization problems in Sect. 7 and outline directions for future research in Sect. 8.

2 Definition of tolerances

In this section, we present formal definitions of CMPs and their upper and lower tolerance values. Moreover, we show how individual upper and lower tolerance values can be computed by solving a modified instance of the problem at hand in the case when the objective function of the problem is additive.

Following the definition from [20,23], a *combinatorial minimization problem* CMP $(\mathcal{E}, c, \mathcal{D}, f_c)$ is a problem of finding

$$S^* \in \arg\min\{f_c(S) \mid S \in \mathcal{D}\},$$

where $c : \mathcal{E} \to \Re$ is a given *instance* of the problem with a *finite ground set* $\mathcal{E}$, $\mathcal{D} \subseteq 2^{\mathcal{E}}$ is a *set of feasible solutions*, and $f_c : \mathcal{D} \to \Re$ is an *objective function* of the problem. The set of all optimal solutions is denoted by $\mathcal{D}^* = \arg\min\{f_c(S) \mid S \in \mathcal{D}\}$. It is assumed that $\mathcal{D}^* \neq \emptyset$ and that $S \neq \emptyset$ for some $S \in \mathcal{D}$.

The definition of tolerances is then as follows; see also [20]. We modify the given instance c such that the cost of a single element is increased or decreased. Let $e \in \mathcal{E}$ and $\alpha \geq 0$. By $c_{\alpha,e} : \mathcal{E} \to \Re$ we denote the instance defined as $c_{\alpha,e}(a) = c(a)$ for each $s \in \mathcal{E}\backslash\{e\}$, and $c_{\alpha,e}(e) = c(e) + \alpha$. Let S^* be an arbitrary element from $\mathcal{D}^*$. The *upper tolerance of e with respect to S^**, denoted by $u_{S^*}(e)$, is defined as $u_{S^*}(e) = \sup\{\alpha \geq 0 : S^* \in \arg\min\{f_{c_{\alpha,e}}(S) : S \in \mathcal{D}\}\}$ and the *lower tolerance of e with respect to S^**, denoted by $l_{S^*}(e)$, as $l_{S^*}(e) = \sup\{\alpha \geq 0 : S^* \in \arg\min\{f_{c_{-\alpha,e}}(S) : S \in \mathcal{D}\}\}$ [20]. Clearly, $u_{S^*}(e') = +\infty$ and $l_{S^*}(e'') = +\infty$ for any $e' \in \mathcal{E}\backslash S^*$ and $e'' \in S^*$. Hence, if $e \in S^*$, then $u_{S^*}(e)$ is the maximal increase of $c(e)$ under which S^* stays optimal, provided that the cost coefficients of all other elements remain unchanged. Similarly, if $e \in \mathcal{E}\backslash S^*$ and $l_{S^*}(e)$ is the maximal decrease of $c(e)$ under which S^* stays optimal, provided that the other parameter values remain unchanged. In this paper, we assume that we have a fixed optimal solution S^* of our problem and drop the subscript S^* for tolerances. We write $u(e)$ and $l(e)$ instead of $u_{S^*}(e)$ and $l_{S^*}(e)$ and consider only finite tolerances.

If the objective function is additive, it holds, according to Theorem 4 from [20], that the upper tolerance value $u(e)$ of an element e in the selected optimal solution S^* can be computed as follows:

$$u(e) = f_c(S^*_-(e)) - f_c(S^*), \tag{1}$$

where $S^*_-(e)$ is the best solution from which e is forbidden. Likewise, if the cost function is additive, then the lower tolerance $l(e)$ of an element e outside of S^* can be computed as

$$l(e) = f_c(S^*_+(e)) - f_c(S^*) \tag{2}$$

according to Theorem 11 from [20], where $S^*_+(e)$ denotes the best solution in which e must be included. The property in Eq.(1) and (2) also holds for some other types of objective functions, such as the bottleneck objective; see [22].

The definition of upper and lower tolerances and the results in Eq. (1) and (2) can be extended easily to Combinatorial Maximization Problems with additive objective functions, such as the WIST. The upper and lower tolerance values then correspond to the *profit decrease* from excluding or including elements, respectively.

The optimal solution $S^*_-(e)$ can be obtained by solving an instance c' such that $c'(a) = c(a)$ for every $a \in \mathcal{E}\backslash\{e\}$ and $c'(e) = \infty$. To compute the lower tolerance, we determine $S^*_+(e)$ by including the requested element e in the CMP by setting its value to $-\infty$.

The consequence of the reported results is that in case of some objective functions, such as additive ones, upper and lower tolerance values of CMPs can be computed by solving a modified problem instance. However, as we will see in the following sections, solving such modified instances can be ineffective, if we wish to determine many tolerance values.

3 The usage of tolerances in solution methods

In this section, we summarize tolerance-based algorithms for CMPs and illustrate that they can be fast, effective, and accurate. We show the usage of tolerances in existing algorithms and we discuss how many tolerances should be computed within a given method.

As shown in Sect. 2, a tolerance value gives the increase in objective value if an element is excluded from or included into the current solution if the objective function is additive. In that sense, any method that computes the costs of candidate solutions obtained by forbidding a single element or forcing a single element into it before deciding on the candidate can

be considered a tolerance-based method, even when the concept of tolerances is not used explicitly. This applies to the methods from [15,28], and [44].

The first implicit application of tolerances is, to the best of our knowledge, the well known procedure from [44] for the *transportation simplex algorithm*. In this method, the difference between the second lowest and the lowest cost in each row and column of the cost matrix is computed and used. In the RAP, the purpose is to find the set of row minima: the difference between the second smallest and the smallest value is therefore an upper tolerance value of the RAP; see Sect. 7.

The *modified Lin–Kernighan heuristic (LKH)* from [28] solves the STSP by solving a so-called 1-Tree Problem (1-TP) relaxation. It then takes connections from outside of the 1-TP solution into consideration for inclusion in candidate solutions if their so-called α-*'nearness'* values are sufficiently low. Such a α-nearness value is defined as the cost increase if the connection is included in a 1-TP solution and is therefore a lower tolerance value. This tolerance-based method has been improved in [45].

Many applications of tolerance-based algorithms have used the AP relaxation. The papers by [17] and [51] introduce tolerance-based branch and bound methods for the ATSP: the former uses only upper tolerances and the latter one uses lower tolerances as well. Heuristics for the ATSP from [19,21] also use AP tolerances; see also [18]. The algorithms in [2] and [4] are able to improve the search for exact solutions to the asymmetric distance-constrained vehicle routing problem (ADVRP) and asymmetric capacitated vehicle routing problem (ACVRP), respectively.

Some studies use unconventional relaxations and their tolerances to solve less well-known problems. The methods from [5,6] solve the problem of finding minimum weight matchings in order to determine the roots of a system of polynomial equations and use the RAP relaxation and tolerances to that end. Some of the heuristics for the ATSP in [19], such as 'R-R-GREEDY', use the RAP tolerances as well. In the paper by [25], the WIST relaxation and tolerances are used iteratively in a heuristic for finding solutions to the WISP; see Sect. 6.

Note that the usage of tolerances is quite similar to the usage of *strong branching* for ILP problems; see [1]. In strong branching, one should select an integer variable with non-integer value in the current LP solution for branching. To that end, two new LP problems are solved for each of these variables: one where the variable's value should be larger than or equal to the current value rounded up and one where it is should be smaller than or equal to the current value rounded down. The integer variable selected for branching is the one for which the increase in solution values is the largest (in a minimization problem).

The usage of tolerances has led to algorithms that are faster or provide better solutions than existing algorithms. The algorithms from [2] solve ADVRP instances that cannot be solved with CPLEX and have shorter solution times for the instances that CPLEX can solve. The LKH heuristic has long been considered one of the best heuristics for the STSP and it has been improved further in [45] to determine new best solutions to yet unsolved instances. The LBnB method presented in [17] returns with a smaller gap than competitors for the challenging ATSP instance `super316.10`. Tolerance-based methods are also used to solve novel problems such as the WISP [25].

The role of tolerances in these methods is twofold. Firstly, tolerances serve to find likely candidates to be in a good or optimal solution to the original problem. In heuristics, tolerances can be used to construct a candidate set of elements, as in [28], or to decide which elements to keep or preserve in each iteration, as in [25]. In branch and bound algorithms, they can be used to determine which element should be included or excluded in a branching step. With upper tolerances, we can determine which elements are preserved [51], but if we include lower tolerances, we can strengthen the analysis by determining which elements should not

be selected, as in [45]. Secondly, tolerances can be used to construct lower bounds on the optimal solution value of the original problem. For example, the ADD-DISJ procedure has been constructed for that purpose and the paper by [17] strengthens a tolerance-based lower bound for the ATSP by including the cost increase from adding connections between cycles using lower tolerances and uses this lower bound at every node of the search tree.

The tolerance-based methods differ in the number of computed tolerances, ranging from a few upper tolerances to all upper and lower tolerances, and in the frequency of the computation of tolerances, ranging from only in the initial step to in each iteration. In particular, if many tolerances are computed during every iteration of an algorithm, it is important that these computations are efficient. Alternatively, frequent and many tolerances computations can only pay off if the methods for computing tolerances are efficient. We now discuss and evaluate the computation of tolerances for some of the most frequently used problems, namely the AP and the MSTP, and for the less frequently considered WIST.

4 Computation of all tolerances of the assignment problem

The assignment problem (AP) is the problem of assigning n workers to n jobs against the minimum cost such that each worker performs exactly one job and each job is performed by exactly one worker. An n by n cost matrix specifies how much it costs to assign each worker to each job. The AP is a popular relaxation for several problems such as the ATSP [30] and asymmetric variants of the Vehicle Routing Problem [2]. The upper and lower tolerances of the AP have been used for these problems in the approaches by, among others, [2,4,17,22,51]. In this section, we explain and compare methods for computing upper and lower tolerances and determine the time needed for each of them in numerical experiments.

For an AP instance, we can draw a complete graph $G = (V, E, c)$ in which the vertices in V represent all workers, the arcs in E represent all possible assignments of workers to jobs, and c is the cost matrix of assigning workers to jobs. Arcs are used to model that the cost of assigning i to j may not be equal to that of j to i; thus, an asymmetric graph is needed. A *cycle* or *tour* in a graph is a sequence of its vertices such that the first and the last elements of the sequence coincide.

Every AP solution can be represented as a set of disjoint cycles; for example, the cycle $(1, 2, 1)$ represents the assignment of the worker 1 to the job 2 and the worker 2 to the job 1. An optimal AP solution is therefore also known as a *minimum cycle cover*.

The LP formulation of the AP is as follows (see, e.g., [37]):

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3}$$

$$\text{s.t.} \sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \tag{4}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall j \in V \tag{5}$$

$$x_{ij} \geq 0 \qquad \forall i, j \in V \tag{6}$$

The solution of an AP instance takes $\mathcal{O}(n^3)$ time using the Hungarian algorithm [36], though faster algorithms exist for sparse instances, i.e., instances with the number of arcs substantially smaller than n^2. An overview over AP codes and solvers is given by Dell'Amico and Toth in [11].

We use the following AP example to illustrate the computation of upper and lower tolerances:

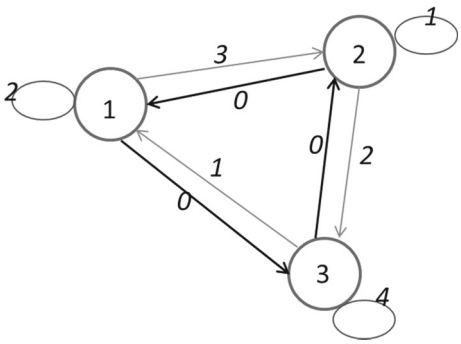


Fig. 1 An AP instance with solution in a *dark color*

Example 1 Consider the following AP instance with 3 vertices (Fig. 1). The arrows represent the connections and the costs are presented near arcs. The loops from each vertex to itself are given by the ellipses.

The cost matrix c is:

$$\begin{pmatrix} 2 & 3 & 0 \\ 0 & 1 & 2 \\ 1 & 0 & 4 \end{pmatrix}$$

In this example, loops exist from each node $i = 1, 2, 3$ to itself. The optimal AP solution consists of the arcs (1, 3), (3, 2), and (2, 1) with the cost 0 and forms the cycle (1, 3, 2, 1).

Upper tolerance values are relevant for arcs inside the selected optimal solution, and lower tolerances are relevant for arcs outside of it. The other upper and lower tolerance values are equal to $+\infty$. This leaves n upper tolerance values and $n(n-1)$ lower tolerance values to be computed.

A naive way of computing the upper tolerance value of an arc $e \in E$ is to create an instance c' such that $c'(e) = \infty$ and $c'(e') = c(e')$ for every $e' \in E \backslash \{e\}$, solve that instance, and subtract the solution value of the original instance from the new one. Similarly, the lower tolerance value of an arc e can be found by solving a new instance in which the arc is forced into a solution and subtracting the solution value of the original instance from that of the new one. If we were to compute all n upper tolerance values of arcs inside the solution and $n(n-1)$ lower tolerances of arcs outside of the solution, the computation of all upper tolerances would take $\mathcal{O}(n^4)$ time, and the computation of all lower tolerances would take $\mathcal{O}(n^5)$ time.

An improvement is that we can compute an upper or lower tolerance value faster if an arc from outside of an optimal AP solution is included into it, or if an arc from the AP solution is excluded from it. Such a modified AP instance can be solved in $\mathcal{O}(n^2)$ time by performing an augmenting path computation [3,33]. Using these computations, upper tolerance values of all arcs inside an optimal AP solution can be computed in $\mathcal{O}(n^3)$ time and lower tolerances of the remaining arcs in $\mathcal{O}(n^4)$ time.

In Example 1, one can check that in order to compute the upper tolerance of the arc (1, 3), we would have to solve the AP instance with the cost matrix

$$\begin{pmatrix} 2 & 3 & \infty \\ 0 & 1 & 2 \\ 1 & 0 & 4 \end{pmatrix}$$

The optimal AP solution of this instance contains the arcs (1, 1), (2, 3), (3, 2) and has the cost 4. Since the original AP solution value is equal to 0, the value of $u(1, 3) = 4 - 0 = 4$.

A further improvement can speed up the computation of the lower tolerances. The paper [53] argues that a procedure in the paper [34] for computing the so-called *hard dual values* of the AP can easily be adapted to compute upper and lower tolerance values. With this procedure, all lower tolerance values can be computed in $\mathcal{O}(n^3)$ time, which is the same complexity as the computation of upper tolerances of all arcs in an optimal AP solution.

Normally, optimal dual values are computed in the dual formulation of an LP problem. For the AP, we have the the dual values u_i and v_i for each $i \in V$ representing the price of leaving and entering a node i, respectively, such that each node is entered and left exactly once, and the values of w_{ij}, the so-called *reduced costs* (not to be confused with the reduced costs in the LP context). The dual formulation of the AP is as follows [34]:

$$\max \sum_{i \in V} (u_i + v_i) \tag{7}$$

$$\text{s.t.} \quad u_i + v_j + w_{ij} = c_{ij} \forall i, \quad j \in V \tag{8}$$

$$w_{ij} \geq 0 \forall i, \quad j \in V \tag{9}$$

The motivation for computing hard dual values in [34] is the following. In the sensitivity analysis of a given optimal AP solution, the LP given in Eq. (3)–(5) is degenerate, meaning that more constraints in the primal formulation are binding than is necessary to define a corner point solution of the AP. Thus, multiple bases can correspond to a single AP solution. If the cost coefficient of an arc is changed, it is possible that this increase leads to a change in the basis, but not to a change in the solution. As a consequence, it is not possible to determine the decreases and increases in each cost coefficient for which the current optimal AP solution remains optimal. In order to resolve this issue, the paper [34] wishes to determine the maximum cost decrease of each arc outside of the current AP solution and the maximum cost increase of each arc in the current AP solution such that the current solution remains unchanged. The resulting values are called *hard dual values*.

Volgenant [53] then shows that the hard dual value of each arc outside of an optimal AP solution is the maximum decrease in the cost value needed to obtain a solution with that arc in it (i.e., the lower tolerance value of the arc). He also shows that the hard dual value of each arc inside the current optimal AP solution coincides with its upper tolerance value.

The hard dual values are computed simultaneously as follows. The *reduced cost matrix* w is an n by n matrix with the elements w_{ij}, as in Eq. (8). Let A^* be an optimal solution of the original AP. The hard dual value $h(i, j)$ of an arc $(i, j) \notin A^*$ is determined by computing the shortest path between i and j in the graph $G' = (V, E, w)$ by [53]. The following numerical example shows this computation of hard dual values for an AP instance with $n = 3$. This procedure is implemented in the lower tolerance-based algorithms in [17].

Example 2 We illustrate the computation of hard dual values with the AP example from Example 1. The cost matrix c is already a reduced cost matrix, so that the matrix w is equal to the matrix c, meaning that $w_{ij} = c_{ij}$ for each $(i, j) \in E$. We compute the hard dual values (which are equal to lower tolerance values) for the arcs out of the vertex 1 not belonging to the optimal AP solution.

The optimal AP solution consists of the assignment from 1 to 3, from 3 to 2, and from 2 to 1, which we write as: $a(1) = 3, a(2) = 1, a(3) = 2$. Finally, a^{-1} is the inverse of a, such that $a^{-1}(i) = j$ for $a(j) = i$ (in other words, $a^{-1}(i)$ is the predecessor of i in the AP solution).

The direct computation of the lower tolerance value of an arc subtract the current optimal AP solution value from the value of the cheapest AP solution that contains the arc. The cheapest solution that includes (2, 2) and (3, 1) is formed by the cycles (2, 2) and (1, 3, 1) with the cost 2, giving $l(2,2) = l(3,1) = 2$; the cheapest solution that includes (1, 1) and (2, 3) is formed by the cycles (1, 1) and (2, 3, 2) with the cost 4, giving $l(2,2) = l(2,3) = 4$, the cheapest solution that includes (1, 2) consists of the cycle (1, 2, 3, 1) with the cost 6, giving $l(1,2) = 6$. Finally, $l(3,3) = 7$ is the cheapest solution with (3, 3) consists of (3, 3), (1, 2, 1) with the cost 7.

We illustrate the computations of $l(1,1)$ and $l(1,2)$ as implemented in the code by [17] and leave the remaining computations to the reader. In order to find these hard dual values, an auxiliary matrix W should be constructed as follows. An (i, j)-entry in the matrix is obtained by taking $W(i,j) = w(j, a(i))$. This is the cost of going from j to $a(i)$. For example, $W(1,1) = w(1, a(1)) = w(1,3) = 0$, $W(1,2) = w(2, a(1)) = w(2,3) = 2$, and $W(1,3) = w(3, a(1)) = w(3,3) = 4$. The matrix W thus becomes:

$$\begin{pmatrix} 0 & 2 & 4 \\ 2 & 0 & 1 \\ 3 & 1 & 0 \end{pmatrix}$$

We define the shortest path through this auxiliary matrix from i to j as $s(i, j)$. The shortest path from 1 to 1, $s(1,1)$, has the length 0, $s(1,2)$ has the length 2, and $s(1,3)$ has the length 3 (through the node 2). This suffices for the computation of the hard dual values of all arcs leaving 1 as we will see below. It holds that $h(i,j) = W(a^{-1}(j), i) + s(i, a^{-1}(j))$ for each $(i, j) \in A^*$ by [34], and $l(i,j) = h(i,j)$, so $l(1,1) = W(2,1) + s(1,2) = 2 + 2 = 4$, and $l(1,2) = W(3,1) + s(1,3) = 3 + 3 = 6$.

The intuition behind this procedure from [34] is that if one adds the assignment (i, j), the job assigned to worker i moves to some other job r, the worker assigned to job r is reassigned to s and so on, until some worker takes the job originally assigned to worker j: now a reassignment is obtained. So, if we include (1, 2), the job 3 becomes available. The easy reassignment is to have the worker that was originally assigned to 2, the worker number 3, take over the job 3. The entry $W(1,3)$ represents this direct reassignment in the matrix W. By performing the shortest path computations, we evaluate the costs of further reassignments, e.g., of assigning the worker 2 to the job 3 and assigning the worker 3 to the job 1.

After the completion of the procedure, upper tolerance values are readily available as well: the upper tolerance value of each arc $(i, j) \in A^*$ is equal to the minimum lower tolerance value of any arc leaving i. It is well-known that the shortest paths from a given node to all other nodes can be obtained in $\mathcal{O}(n^2)$ time using Dijkstra's algorithm [12], which can be improved for sparse graphs to $O((m+n)\log n)$ time in graphs with m arcs and to $O(m \log m)$ if all vertices are reachable from the source [10]. It is shown that the lower tolerances of all arcs directed outward of any vertex $v \in V$ can be computed in $\mathcal{O}(n^2)$ time using the same procedure [17]. This is true as only the shortest paths originating from v have to be determined to that end.

Experiments We perform numerical experiments to compare the speed of an AP solver and the computation of upper and lower tolerances. The AP solver is presented in [32] and the code of this solver can be retrieved from [31]. The computation of the upper tolerances is performed by adding the shortest augmenting path whereas the computation of the lower

Table 1 Time consumption in seconds of Jonker and Volgenant's AP solver, of all upper tolerances computation (UT), and of lower tolerances computation (LT)

	Solve AP	Compute UT	Compute LT
`ft` and `ftv` instances size, $n = 39$–101			
Average	0.01	0.01	0.02
Minimum	0.00	0.00	0.01
Maximum	0.02	0.02	0.05
ftv170, $n = 171$	0.10	0.11	0.21
`coin` instances, $n = 100$			
Average	0.03	0.05	0.08
Minimum	0.02	0.04	0.06
Maximum	0.05	0.08	0.11
`coin` instances, $n = 316$			
Average	0.59	0.91	1.50
Minimum	0.37	0.60	0.98
Maximum	0.80	1.24	2.05
Random, $n = 1000$			
Average	11.36	14.48	25.93
Minimum	8.28	11.96	20.71
Maximum	17.54	19.46	35.35

tolerances is performed with the procedure for computing hard dual values described above and also provide upper tolerance values. All procedures run in $\mathcal{O}(n^3)$ time.

We consider instance types of different sizes. The `ft` and `ftv` instances from the TSPLIB ([43], retrievable from [50]) vary in the size from 39 to 171, the `coin` instances from [30] have the size 100 and 316, retrievable on [13]. In addition, we have generated random instances of the size 1000. For each instance type, 10 instances have been generated (except for the 11 `ft` and `ftv` instances). The minimum, maximum, and average computing times are reported. The experiments have been performed on a Dell 2.30 GHz computer with 4 GB RAM using a Windows 7 operating system.

The results of these experiments are reported in Table 1 and show that the lower tolerances computation takes about 1.5–2 times as much time as the upper tolerances computation, which in turn takes about 1.5–2 times as much time as solving the AP from scratch. This does not only hold in the average case but also for the minimum and maximum computation times and even for individual instances. These ratios appear to be consistent across different instance types and sizes.

Note that a naive approach for computing lower tolerance values solves an AP instance for each arc outside of the optimal AP solution, of which there are $n(n-1)$. The time consumption is similar to that of solving a modified instance to determine an upper tolerance value. The total upper tolerances computation time in Table 1 is for solving n of these modified instances. Therefore, it can be expected that the computation of all lower tolerances would require about $n-1$ times as much time as the times reported in the column 'Compute UT'. This implies a computation time of over 14,000 s for $n = 1000$.

Even though the AP upper and lower tolerances computation only takes 1.5–3 times as much time as the AP computations solved from scratch, the AP computations take much less time if we already have an optimal solution and remove or include a single arc. In that case, computing all upper tolerances takes n times as much time as solving such a modified AP.

5 Computation of all tolerances of the minimum spanning tree problem

A *spanning tree* of a connected graph is a tree obtained by deleting some (perhaps no) edges of the graph, where an edge is an undirected connection between two vertices. The minimum spanning tree problem (MSTP) is a classical network problem with many applications [26]. Given a connected undirected graph $G = (V, E, c)$ with an edge set E, a vertex set V containing n vertices, and a cost function $c : E \to \mathbb{R}$, the *MSTP* is the problem of determining a spanning tree with the minimum sum of weights of edges. A solution of an MSTP instance is called a *minimum spanning tree* or *MST*.

The minimum 1-tree problem (1-TP) is a closely related problem to the MSTP. Let $v \in V$ be a chosen vertex with degree at least two such that the graph G'_v, obtained by deleting v from G, is connected. A *1-tree with respect to* v is defined as follows: find in G the two shortest edges adjacent to v, say e_1 and e_2, and determine an MST for G'_v. The cost of the 1-tree is formed by the cost of the MST on G'_v plus the weights of e_1 and e_2. For a chosen vertex v, the *1-TP* is to find a 1-tree with respect to v. Note that the 1-TP upper tolerance values of all edges of G'_v are simply the MSTP tolerances on the graph G'_v.

The MSTP can be used as a relaxation for $\mathcal{NP}$-hard variations of the MSTP, for example, the degree-constrained MSTP, where the number of MST edges adjacent to each node is limited [52]. The 1-TP is used as a relaxation for the STSP in [27] and [54], and, more recently, in the papers by Helsgaun [28] and Richter et al. [45]. In this section, we explain and compare upper and lower tolerances computation for the MSTP and thereby for the 1-TP.

In the LP formulation of the MSTP, we use the binary decision variables x_e denoting whether an edge e is selected or not, for all $e \in E$. For a subset $S \subseteq V$, let $E(S)$ denote the set of all edges whose both endpoints are in S and let $x(S) = \sum_{e \in E(S)} x_e$. The following LP formulation can be derived from [46]:

$$\min \quad \sum_{e \in E} c_e x_e \tag{10}$$

$$\text{s.t. } x(V) = |V| - 1 \tag{11}$$

$$x(S) \leq |S| - 1 \quad \forall S \subset V, S \neq \emptyset \tag{12}$$

$$x_e \geq 0 \quad \forall e \in E \tag{13}$$

The constraint (11) requires that there are exactly $n - 1$ edges in a solution and the constraint (12) prevents the usage of cycles in a solution. The constraints (11) and (12) jointly ensure that each feasible solution is a spanning tree. Note that there are exponentially many constraint of type (12).

The MSTP can be solved in $\mathcal{O}(n^2)$ time using Prim's algorithm [41] and in $\mathcal{O}(m \log n)$ using Kruskal's algorithm [35], where $m = |E|$. There are faster algorithms for sparse instances, such as the algorithm in [40], that can solve the MSTP in almost linear time in the number of edges m.

We use the following MSTP example to explain the computation of tolerance values.

Example 3 Consider the MSTP example with four vertices and the costs, presented by the following cost matrix c:

The optimal solution (denoted by T^*) consists of the edges (1, 3), (3, 4), and (2, 4) and has the cost 9.

The upper tolerance of an edge e in an optimal MSTP solution T^* can be computed using the result in Eq. 1, and the lower tolerance can be computed using Eq. (2). In both cases, a new MSTP instance is solved.

From/to	1	2	3	4
1	0	5	3	7
2	5	0	8	2
3	3	8	0	4
4	7	2	4	0

However, the MSTP has some properties that make it unnecessary to solve a new MSTP instance from scratch. The *cutset property*, proved in [48], is the following. If an edge e_1 is removed from the current solution, then the remaining edges from the MSTP solution form two disjoint trees. One can then construct the MST without e_1 by including the cheapest edge, denoted by e_2, between these two trees. The upper tolerance value of e_1 would be $u(e_1) = c(e_2) - c(e_1)$. If we remove $(3, 4)$ with the weight 4 in Example 3, the remaining MST edges constitute a subtree on the vertices 1 and 3 connected by $(1, 3)$ and another subtree on the vertices 2 and 4 connected by $(2, 4)$. If we exclude $(3, 4)$, the cheapest connection between those two trees is formed by $(1, 2)$ with the cost 5. Thus, $u(3, 4) = 5 - 4 = 1$. Similarly, if an edge e_1 outside of the selected optimal solution is included, a cycle is formed and a new optimal solution with e_1 can be found by removing the most expensive MST edge from the cycle. For example, if we include $(1, 4)$ with the weight 7 in the example, we obtain the cycle consisting of the edges $(1, 3)$, $(3, 4)$, and $(1, 4)$. Among the two current MSTP edges $(1, 3)$ and $(3, 4)$, $(3, 4)$ with the cost 4 is the most expensive one, so the lower tolerance of $(1, 4)$ can be computed as $l(1, 4) = c(1, 4) - c(3, 4) = 7 - 4 = 3$.

When the upper and lower tolerance values of all edges inside and outside of T^* are computed, it is not necessary to repeat the computations above for each edge. Methods for performing the computations of all tolerances efficiently are described in [9,28,39,48].

Firstly, we describe the method by Chin and Houck [9] to compute all *upper tolerances*. It runs in $\mathcal{O}(n^2)$ time and has $\mathcal{O}(n)$ memory requirements. The authors include a detailed pseudo-code of the computations in their paper. We illustrate the main computations of this approach here and refer to the original paper for the source code.

The procedure determines the labels $l_e(r, s)$ of the edge e, i.e., the nodes $v, w \in V$, such that (v, w) comes into the solution if (r, s) is removed; we then write $l_e(r, s) = v, w$. The upper tolerance of each edge (r, s) is computed as $u(r, s) = c(v, w) - c(r, s)$. The procedure starts with an arbitrary vertex $r \in V$. Then, it explores in a depth-first search fashion the nodes in the MST, meaning that we explore a branch of the tree entirely before proceeding to the next branch.

For each newly considered vertex v, we consider the edges towards the already considered vertices. If an edge from e to one of the already considered vertices is cheaper than a currently cheapest alternative, this edge replaces that alternative and the label of this edge are then updated. We continue iteratively in the described depth-first search fashion until all vertices have been considered. We illustrate the procedure below.

Example 4 Assume that we start at the node 1 with the data given in Example 3. As mentioned before, we traverse the MST in a depth-first search fashion. In this case, this is relatively easy as there is one branch: we traverse the edges in the order $(1, 3)$, $(3, 4)$, and $(2, 4)$.

The (only) adjacent vertex of 1 in the MST is 3. For this vertex, we determine the set of adjacent vertices: $\{1, 4\}$. The adjacent vertex 1 is uninteresting, but we should consider vertex 4 and its connections to all previously considered vertices, namely 1. The edge $(1, 4)$, with the cost value 7, is now the cheapest way of connecting the components obtained by removing $(1, 3)$ or $(3, 4)$. The labels of $(1, 3)$ and $(2, 4)$ are, in both cases, set to 1, 4.

From vertex 3, we proceed to vertex 4, which is adjacent to 2 and 3 in the MST (where 3 has already been considered). The vertex 2 can be connected to the previously encountered vertex 1 and 3. The first vertex that we encounter when we backtrack to our source node 1 is vertex 3. The edge (2, 3) has cost 8 and is more expensive than the current alternative to (1, 3) and (2, 4), which is (1, 4). However, this is currently the cheapest alternative to (3, 4); thus, we set the labels of (3, 4) to 2, 3.

The edge (2, 1) has the cost 5, which is cheaper than that of both (1, 4) and (2, 3), meaning that the cheapest way of reconnecting the components obtained from removing (1, 3), (3, 4) and (4, 2) is by adding (2, 1). The labels of the edges (1, 3) and (3, 4), currently labeled 1, 4, and of the edge (1, 2), labeled 2, 3, should be set to 1, 2. Thus we have the labels 1, 2 for all three edges, giving the upper tolerances $u(1,3) = c(1,2) - c(1,3) = 5 - 3 = 2$, $u(3,4) = 5 - 4 = 1$, and $u(4,2) = 5 - 2 = 3$.

A quite detailed pseudo-code of the procedure can be found in the Algorithm DE ('deletion of edge') in [9].

Helsgaun [28] presents a procedure for computing *all lower tolerance values* in $\mathcal{O}(n^2)$ time with $\mathcal{O}(n^2)$ memory. Actually, the approach is applied to the 1-TP Problem, but the tolerances of the 1-TP are largely lower tolerance values of the MSTP. The approach determines an α-nearness value of an edge, the maximum decrease in the cost of an edge before it enters into an optimal MSTP solution, which equals the lower tolerance value of the edge with respect to the solution. These α-nearness values are used in [28] in the so-called LKH heuristic in order to reduce the set of candidate edges for selection; see also Sect. 3.

Two versions of the procedure for computing the α-nearness values are described in the pseudo-code in Figures 7 and 8 of [28]. Here, we adapt the code in Figure 7 to the MSTP and present it below. The function dad(i) outputs the predecessor of a vertex i in the MST. An arbitrary node could be selected as a root node. We select the node with index 1.

Input:
A vertex set $V = \{1, 2, \ldots, n\}$ and an edge set E
A cost function $c : E \to \mathbb{R}$
An MST T^*, given by the function dad$(\cdot)$.
Computations:
for $i = 1$ to n **begin**
$\beta(i, i) = 0$;
for $j = i + 1$ to n
$\beta(i, j) = \beta(j, i) = \max(\beta(i, \text{dad(i)}), c(j, \text{dad(i)}))$;
end
end
Output:
The lower tolerance value $l(i, j) = c(i, j) - \beta(i, j)$ for each $(i, j) \notin E(T^*)$.

The computation of $\beta(i, j) = \beta(j, i) = \max(\beta(i, \text{dad(i)}), c(j, \text{dad(i)}))$ requires the value of $\beta(i, \text{dad(i)})$, which in turn requires the computation of $\beta(i, \text{dad(dad((i)))}$, and so on. In practice, we compute these values during the computation of $\beta(i, j)$ if they have not been computed previously, as we illustrate using the running example.

Example 5 In the MSTP from Example 3, we use the node 1 as the root node. Then, dad(3) = 1, dad(4) = 3, and dad(2) = 4. We illustrate the computation of $\beta(1, 4)$: $\beta(1,4) = \max(\beta(1, \text{dad}(4)), c(4, \text{dad(i)}) = \max(\beta(1,3), c(4,3))$. Then we have to determine $\beta(1,3) = \max(\beta(1, \text{dad}(3)), c(3, \text{dad}(3)) = \max(\beta(1,1), c(1,3)) = \max(0, 3) = 3$.

So, $\beta(1,4) = \max(3,4) = 4$. That means that $l(1,4) = 7 - 4 = 3$. All lower tolerance values are computed in a similar vein and the results are $l(1,2) = 1, l(1,4) = 3, l(2,3) = 4$.

There are also procedures for computing all upper and lower tolerance values jointly, which are particularly effective for relatively sparse matrices. Those methods are more difficult to implement, as the papers in which the procedures are presented often do not contain codes. Tarjan [48] introduces a procedure for computing all upper and lower tolerance values that runs in $\mathcal{O}(\alpha(m,n)m)$ time, where the term $\alpha(m,n)$ is the very slowly growing inverse Ackermann function and m is the number of edges. The memory requirement of this procedure is $\mathcal{O}(m)$. Improvements to the procedure by [48] have been presented in the work by Pettie et al. [39], where the authors show that their procedure runs in $\mathcal{O}(mlog(\alpha(m,n))$ time. This efficiency increase is achieved by implementing advanced data structures. The method for solving the MSTP presented in [40] has the complexity $\mathcal{O}(mlog(\alpha(m,n))$. Booth and Westbrook [7] present an $\mathcal{O}(m)$ algorithm for the sensitivity analysis of minimum spanning trees in planar graphs with $m = O(n)$. The paper by Dixon et al. [14] introduces a method that uses $\mathcal{O}(m)$ time multiplied by a large constant.

Experiments How much time does it take to perform computations of all upper and lower tolerances in comparison to the MSTP solution time? We have performed numerical experiments on complete weighted graphs. In Table 2, we report the resulting computation times. We use the method of Chin and Houck [9] for computing all lower tolerances, the method of Helsgaun [28] for computing all upper tolerances, and Prim's algorithm [41] for determining an optimal MSTP solution.

We consider randomly generated instances of the size 1280 and 2560. The first class is formed by Euclidean instances with uniformly distributed coordinates are generated in $\mathbb{R}^2$ in a plane measuring 100 by 100. The second class contains instances with distances in a symmetric distance matrix, where the (integer) distances are also uniformly generated between 1 and 10,000. For each instance type, 10 instances have been generated for which the minimum, maximum, and average computing times are reported. The experiments have been performed on a Dell 2.30 GHz computer with 4 GB RAM using a Windows 7 operating system.

The results show that the computation of both the upper and the lower tolerance requires about 2–4 times as much time as Prim's algorithm. It appears that the computation of all lower tolerances computation does not take more time than that of all upper tolerances.

The tolerances computation and Prim's algorithm appear to take more time for the Euclidean MSTP instances than for the randomly generated distance instances. In Euclidean instances, the coordinates of the locations are stored and distances are computed each time they are needed in order to save memory. If these distances are stored in a cost matrix instead, the difference in computation time disappears. The inclusion of such distance computations does not change the relative time consumption of the upper and lower tolerances computation and Prim's algorithm.

6 Computation of all tolerances of the weighted independent set problem for trees

We have illustrated how upper and lower tolerance values can be computed for the polynomially solvable problems MSTP and AP. However, fast methods for computing tolerances also exist or can be designed for other less frequently used relaxation problems. In this section, we

Table 2 Time consumption in seconds of Prim's MSTP method, of all upper tolerances computation (UT), and of lower tolerances computation (LT)

Find Using	MSTP Prim	UT Chin-Houck	LT Helsgaun
Random, $n = 1280$			
Average	0.13	0.62	0.60
Min.	0.08	0.37	0.36
Max.	0.21	0.93	1.00
Random, $n = 2560$			
Average	0.49	2.10	1.87
Min.	0.28	1.41	1.40
Max.	0.95	3.93	3.81
Random, $n = 1280$, Euclidean			
Average	0.20	0.73	0.68
Min.	0.09	0.36	0.36
Max.	0.33	1.14	1.18
Random, $n = 2560$, Euclidean			
Average	0.90	3.23	2.33
Min.	0.34	1.54	1.61
Max.	1.46	4.90	3.57

illustrate that upper and lower tolerances can be computed fast and jointly for the weighted independent set problem for trees (WIST) based on the results from Goldengorin et al. [24].

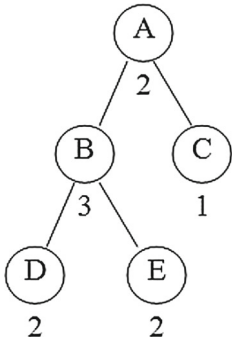
The WIST is used as a relaxation for the general weighted independent set problem (WISP), which is defined for a simple graph with weights on its vertices. The objective is to determine a set of pairwise non-adjacent vertices with the maximum sum of weights. The general problem is $\mathcal{NP}$-hard, but if the given graph is a tree, then the problem, the WIST, can be solved in linear time; see [8].

It is important to note that the WIST is a maximization problem. Therefore, the definitions of upper and lower tolerances are modified in comparison to those given in Sect. 2. Given an optimal solution S^* consisting of a set of vertices, the *upper tolerance of a vertex* $v \notin S^*$ is the largest increase in the weight of this vertex $w(v)$ such that the solution S^* remains optimal. Likewise, the *lower tolerance of a vertex* $v \in S^*$ is the largest decrease in the weight $w(v)$ such that S^* remains optimal. The upper tolerances of vertices $v \in S^*$ and the lower tolerances of vertices $v \notin S^*$ are infinitely large. The WIST has an additive objective function and therefore it is easy to show that the upper and lower tolerance values measure the profit decrease from including $v \notin S^*$ and from excluding $v \in S^*$, respectively.

In the paper by Goldengorin et al. [24], it is shown that the WIST can be solved in $\mathcal{O}(n)$ time and that the simultaneous computation of all upper and lower tolerances has the same time complexity. Below, we provide a numerical example of the solution and the computation of the tolerances of the WIST (for a pseudo-code of the algorithm, we refer to the paper [24]). In the example, we construct an optimal solution S^* of the WIST in the first pass. In the second pass, for each subtree rooted at a vertex v, we determine two solutions: the best solution with v in it and the best solution without v. In the third pass, we use these values to determine upper tolerances of the vertices outside of S^* and lower tolerances of the vertices in S^*.

Example 6 The weighted tree is presented in Fig. 2 above, where the numbers below the vertices denote the vertex weights. For the given tree (A, B, C, D, E) is the breadth-first

Fig. 2 A weighted tree



search order. In order to determine S^*, we pass through the vertices of the graph, starting at the leaves of the tree. Then, we work our way up in the tree, until we reach the top node A. For any vertex $v \in \{A, \ldots, E\}$ and the tree, rooted at v, the term $s(v)$ denotes an optimal solution, $s_{in}(v)$ is the best solution among all solutions containing v, and $s_{out}(v)$ is the best solution among all solutions not containing v. The terms $w(v)$, $w_{in}(v)$, $w_{out}(v)$ denote the weights of $s(v)$, $s_{in}(v)$, $s_{out}(v)$, respectively. Since A is the top node and the search process ends there, it holds that the value of an optimal solution S^* equals to $w(A)$, i.e. $f_c(S^*) = w(A)$.

The first pass is as follows:

1. $w_{in}(E) = c(E) = 2,\ w_{out}(E) = 0,\ s_{in}(E) = \{E\},\ s_{out}(E) = \emptyset, w(E) = 2,\ s(E) = \{E\}$;
2. $w_{in}(D) = c(D) = 2,\ w_{out}(D) = 0,\ s_{in}(D) = \{D\},\ s_{out}(D) = \emptyset,\ w(D) = 2,\ s(D) = \{D\}$;
3. $w_{in}(C) = c(C) = 1,\ w_{out}(C) = 0,\ s_{in}(C) = \{C\},\ s_{out}(C) = \emptyset,\ w(C) = 1,\ s(C) = \{C\}$;
4. $w_{in}(B) = c(B) + w_{out}(D) + w_{out}(E) = 3,\ s_{in}(B) = \{B\} \cup s_{out}(D) \cup s_{out}(E) = \{B\},\ w_{out}(B) = w(D) + w(E) = 4,\ s_{out}(B) = s(D) \cup s(E) = \{D, E\},\ w(B) = \max(w_{in}(B),\ w_{out}(B)) = 4,\ s(B) = \{D, E\}$;
5. $w_{in}(A) = c(A) + w_{out}(B) + w_{out}(C) = 6,\ s_{in}(A) = \{A\} \cup s_{out}(B) \cup s_{out}(C) = \{A, D, E\},\ w_{out}(A) = w(B) + w(C) = 5,\ s_{out}(A) = s(B) \cup s(C) = \{C, D, E\},\ w(A) = \max(w_{in}(A),\ w_{out}(A)) = 6,\ s(A) = \{A, D, E\}$.

Therefore, $S^* = s(A) = \{A, D, E\}$ and $f_c(S^*) = w(A) = 6$.

In the second pass, we iterate from the top node A downwards, again in a breadth-first search fashion, until we reach the end nodes. For each vertex $v \in \{A, B, \ldots, E\}$ and the whole tree, the terms $W_{in}(v)$ and $W_{out}(v)$ denote the values of the best solutions with v and without v, respectively.

1. $W_{in}(A) = w_{in}(A) = 6, W_{out}(A) = w_{out}(A) = 5$;
2. $W_{in}(B) = W_{out}(A) - w(B) + w_{in}(B) = 4, W_{out}(B) = w_{out}(B) + \max(W_{in}(A) - w_{out}(B), W_{out}(A) - w(B)) = 6$;
3. $W_{in}(C) = W_{out}(A) - w(C) + w_{in}(C) = 5, W_{out}(C) = w_{out}(C) + \max(W_{in}(A) - w_{out}(C), W_{out}(A) - w(C)) = 6$;
4. $W_{in}(D) = W_{out}(B) - w(D) + w_{in}(D) = 6, W_{out}(D) = w_{out}(D) + \max(W_{in}(B) - w_{out}(D), W_{out}(B) - w(D)) = 4$;
5. $W_{in}(E) = W_{out}(B) - w(E) + w_{in}(E) = 6, W_{out}(E) = w_{out}(E) + \max(W_{in}(B) - w_{out}(E), W_{out}(B) - w(E)) = 4$.

Similarly to the formulae from Section 2, in order to obtain the tolerances, we should subtract $W_{out}(v)$ and $W_{in}(v)$ from $f_c(S^*)$, respectively. Thus, in the third pass, we have:

1. As $A \in S^*$, $l_{S^*}(A) = f_c(S^*) - W_{out}(A) = w(A) - W_{out}(A) = 1$;
2. As $B \notin S^*$, $u_{S^*}(B) = f_c(S^*) - W_{in}(B) = 2$;
3. As $C \notin S^*$, $u_{S^*}(C) = f_c(S^*) - W_{in}(C) = 1$;
4. As $D \in S^*$, $l_{S^*}(D) = f_c(S^*) - W_{out}(D) = 2$;
5. As $E \in S^*$, $l_{S^*}(E) = f_c(S^*) - W_{out}(E) = 2$;

In general, computing an optimal solution and all tolerances with respect to it takes $\mathcal{O}(n)$ time, assuming that we know the structure of the tree.

The method, presented above for tolerances computation of the WIST, is used in the tolerance-based heuristic for the WISP for a graph G, introduced in [25] as follows. If the graph G is a tree, then the WISP can be solved easily. If G is not connected, then we call the heuristic for each of its connected components. If G is connected, we find a maximum spanning tree T on G, where the weight of each edge is formed by the maximum of the weights of its two endpoints and where the sum of the weights in the tree is maximized. Then, the heuristic determines an optimal WIST solution S^* on the graph $G' = (V, T)$ and determines the upper and lower tolerances of all vertices with respect to the optimal WIST solution. The heuristic then determines the vertex with the maximum upper or lower tolerance value and, if the lower tolerance value is the larger, includes the corresponding vertex in the graph and removes the adjacent vertices or, if the upper tolerance value is the larger, removes the corresponding vertex from the graph. It continues these steps until a feasible WISP solution is found. In each iteration, the computation of tolerances is performed. The whole process takes polynomial time.

It is shown by Malyshev and Pardalos [38] that variants of the WISP on other types of graphs than trees, such as bipartite and interval graphs, can be solved polynomially and that upper and lower tolerance values of these variants can be computed effectively and simultaneously.

7 Computation of all tolerances of some other problems

Shier et al. [47] present an effective computation approach for upper and lower tolerances for the so-called Shortest Path Tree Problem (SPTP), where the purpose is to find a collection of the shortest paths from a given root node to all other nodes. These shortest paths can be represented with a directed tree, the shortest path tree. The computation of upper and lower tolerances of all arcs takes $\mathcal{O}(n^2)$ time. The procedures by [39] and [48] from Sect. 5 can easily be adjusted to determine all upper and lower tolerances jointly for the SPTP in the same order of time.

The results in [47] have been extended to the Shortest Path Problem (SPP), the problem of finding the shortest path between a source node and a sink node in a network without negative cycles. Ramaswamy et al. [42] present a method for computing all upper and lower tolerances in $\mathcal{O}(m + p \log p)$ time, where p is the number of arcs in the shortest path and m is the total number of arcs in the graph instance.

The final problem that we consider is the Relaxed Assignment Problem (RAP), the problem of finding a minimum in each row in an n by n cost matrix. This problem is used as a relaxation for the ATSP in [21]. The RAP upper tolerance value of an element is simply obtained by subtracting its cost from that of the second cheapest element in the row, taking $\mathcal{O}(n)$ time to compute for all upper tolerances assuming that the second-lowest value in each row is

stored, when the RAP is solved. The lower tolerance value of each element can be computed in constant time by subtracting the corresponding row minimum. If we compute all $n^2 - n$ lower tolerances, we use $\mathcal{O}(n^2)$ time. In this problem, the computation times for upper and lower tolerance values thus differ by orders of magnitude.

8 Concluding remarks and future research

In combinatorial approaches for finding exact or approximate optimal solutions, a challenge is often to determine which elements from a current (relaxation) solution should be maintained or discarded. A relatively novel approach is to use tolerances to take such decisions. The challenge with tolerances is that they require computational effort. Thus, in the construction of tolerance-based algorithm, it may be necessary, or desirable, to compute sets of tolerances efficiently. For example, the branch and bound methods from [17,51] require the computation of tolerances at each node of the search tree that can contribute to a large part of the computational burden of such algorithms. In this paper, we present the efficient methods for computing all upper and lower tolerance values for the well-known Assignment Problem (AP) and Minimum Spanning Tree Problem (MSTP) and the less known Weighted independent Set Problem for Trees (WIST). We investigate the reduction in tolerances computation times, if efficient methods are used.

Efficient methods enable us to compute all upper and lower tolerances much more quickly than methods that compute all tolerances individually. Compared to naive approaches that compute each tolerance value individually, the procedure described in [53] reduces the complexity of computing AP lower tolerances from $\mathcal{O}(n^4)$ (or $\mathcal{O}(n^5)$ if the modified APs are solved from scratch) to $\mathcal{O}(n^3)$. The complexity of computing all MSTP upper tolerances can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ using the procedure from [9] and the complexity of computing MSTP lower tolerances is reduced from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^2)$ using the procedure from [28]. Likewise, the computation of the tolerances for the WIST have the same $O(n)$ time complexity as the solution of the problem instead of $O(n^2)$ if tolerances were computed with a naive approach. Our numerical experiments show that these improvements can also be observed in the average time consumption.

In this paper, we compare the time consumption of computing all lower tolerances to that of computing all upper tolerances and the solution of the original problem. For some problems such as the AP and the MSTP, there are up to n times as many elements outside of a solution as inside and there are potentially up to n times as many lower tolerances as upper tolerances. One would expect that it takes much more time to compute all lower tolerance values than it takes to compute all upper tolerance values. However, this is often not true. For the AP and the MSTP, the computation of all lower tolerance values has the same worst time complexity as the computation of all upper tolerance values. Our numerical experiments indicate that it takes slightly more time to compute all lower tolerances for the AP compared to the upper tolerances and about the same amount of time for the MSTP. For some other problems, such as the WIST presented in Sect. 6 and the SPP mentioned in Sect. 7, the fastest methods compute upper and lower tolerances jointly, yielding that both types of tolerances can be obtained with the same complexity. The complexity of computing all upper tolerances is lower than that of computing all lower tolerances for only one of the encountered problems, namely, the RAP.

This paper has presented existing, ready-to-use methods for tolerance-based methods with the AP or the MSTP as a relaxation. The results can also provide guidelines for practitioners

or scientists who wish to develop tolerance-based algorithms with other relaxations. Firstly, one should determine whether, for the problem at hand, efficient methods for computing tolerances exist, for example from the field of determining the sensitivity of optimal solutions (as holds for the AP, the MSTP, and the SPP). It is an advantage if such effective methods can compute upper and lower tolerance values jointly. For some problems, one may have to develop novel efficient methods as has been done for the WIST. Finally, one should decide on which tolerances to use: (all) upper tolerances, (all) lower tolerances, or both types.

An interesting direction of future research is to construct efficient methods for computing upper and lower tolerances computation methods for other relaxations than the one considered in this paper. Another direction of future research is to investigate the usage of tolerances for problems with different objective functions than the additive one such as the bottleneck objective function.

Acknowledgements The research of D.S. Malyshev and P.M. Pardalos is partially supported by LATNA laboratory, National Research University Higher School of Economics. B. Goldengorin's research is supported by C. Paul Stocker Visiting Professorship provided by the Department of Industrial and Systems Engineering, The Russ College of Engineering, Ohio University, Athens, OH, USA.

References

1. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**(1), 42–54 (2004)
2. Almoustafa, S., Hanafi, S., Mladenovic, N.: New exact method for large asymmetric distance-constrained vehicle routing problem. Eur. J. Oper. Res. **226**, 386–394 (2013)
3. Balas, E., Toth, P.: Branch and bound methods. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) Chapter 10 of the Traveling Salesman Problem, pp. 361–401. Wiley, Chichester (1985)
4. Batsyn, M., Goldengorin, B.I., Kocheturov, A., Pardalos, P.M.: Tolerance-based versus cost-based branching for the asymmetric capacitated vehicle routing problem. In: Goldengorin, B.I. et al. (eds.) Models, Algorithms, and Technologies for Network Analysis. Proceedings in Mathematics and Statistics, vol. 59, Springer, Berlin (2013)
5. Bekker, H., Braad, E.P., Goldengorin, B.: Using bipartite and multidimensional matching to select the roots of a system of polynomial equations. Lect. Notes Comput. Sci. **3483**, 397–406 (2005)
6. Bekker, H., Braad, E.P., Goldengorin, B.: Selecting the roots of a small system of polynomial equations by tolerance-based matching. Lect. Notes Comput. Sci. **3503**, 610–613 (2005)
7. Booth, H., Westbrook, J.: A linear algorithm for analysis of minimum spanning and shortest-path trees of planar graphs. Algorithmica **11**, 341–352 (1994)
8. Chen, C., Kuo, M., Sheu, J.: An optimal time algorithm for finding a maximum weight independent set in a tree. BIT Numer. Math. **28**, 353–356 (1988)
9. Chin, F., Houck, D.: Algorithms for updating minimal spanning trees. J. Comput. Syst. Sci. **16**, 333–344 (1978)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Section 24.3: Dijkstra's Algorithm in Introduction to Algorithms, 2nd edn, pp. 595–601. MIT Press and McGraw Hill, NewYork (2001)
11. Dell'Amico, M., Toth, P.: Algorithms and codes for dense assignment problems: the state of the art. Discrete Appl. Math. **140**, 1–3 (2004)
12. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)
13. DIMACS, Benchmark instance generates for the ATSP. http://dimacs.rutgers.edu/Challenges/TSP/atsp.html (2006)
14. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J. Comput. **21**(6), 1184–1192 (1992)
15. Fischetti, M., Toth, P., Vigo, D.: A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper. Res. **42**(5), 846–859 (1994)
16. Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems. Manag. Sci. **27**(1), 1–18 (1981)
17. Germs, R., Goldengorin, B., Turkensteen, M.: Lower tolerance-based branch and bound algorithms for the ATSP. Comput. Oper. Res. **39**(2), 291–298 (2012)

18. Ghosh, D., Goldengorin, B., Gutin, G., Jäger, G.: Tolerance-based algorithms for the traveling salesman problem. In: Neogy, S.K., Bapat, R.B., Das, A.K., Parthasarathy, T. (eds.) Mathematical Programming and Game Theory for Decision Making. Statistical Science Interdisciplinary Research, 47–59, World Scientific Publication, Hackensack, NJ (2008)
19. Ghosh, D., Goldengorin, B., Gutin, G., Jäger, G.: Tolerance-based algorithms for the traveling salesman problem. In: Neogy, S.K., Bapat, R.B., Das, A.K., Parthasarathy, T. (eds.) Chapter of Mathematical Programming and Game Theory for Decision Making, pp. 47–59. World Scientific, New Jersey (2008)
20. Goldengorin, B., Jäger, G., Molitor, P.: Some Basics on Tolerances. In: Cheng, S.-W., Poon, C.K. (eds.) Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM). Lecture Notes in Computer Science **4041**, 194–206 (2006)
21. Goldengorin, B., Jäger, G. Molitor, P.: Tolerance-based Contract-or-Patch Heuristic for the Asymmetric TSP. In: Erlebach, T. (ed.) Proceedinggs 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN). Lecture Notes in Computer Science **4235**, 86–97 (2006)
22. Goldengorin, B., Sierksma, G., Turkensteen, M.: Tolerance-based algorithms for the ATSP. In: Hromkovic, J., Nagl, M., Westfechtel, B. (eds.) Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG). Lecture Notes in Computer Science **3353**, 222–234 (2004)
23. Goldengorin, B., Sierksma, G.: Combinatorial Optimization Tolerances Calculated in Linear Time. Research Report 03A30, Graduate School/Research Institute Systems, Organizations and Management, University of Groningen, Groningen, The Netherlands (2003)
24. Goldengorin, B., Malyshev, D.S., Pardalos, P.M.: Efficient computation of tolerances in weighted independent set problems for trees. Dokl. Math. **87**(3), 368–371 (2013)
25. Goldengorin, B., Malyshev, D.S., Pardalos, P.M., Zamaraev, V.A.: A tolerance-based heuristic for the weighted independent set problem. J. Comb. Optim. **29**, 433–450 (2015)
26. Graham, R.L., Hell, P.: On the history of the minimum spanning tree problem. Ann. Hist. Comput. **7**(1), 43–57 (1985)
27. Held, M., Karp, R.: The traveling-salesman problem and minimum spanning trees. Oper. Res. **18**, 1138–1162 (1970)
28. Helsgaun, K.: An effective implementation of the Lin–Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
29. Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research, 6th edn. McGraw Hill, Singapore (1995)
30. Johnson, D.S., Gutin, G., McGeoch, L.A., Yeo, A., Zhang, W., Zverovich, A.: Experimental analysis of heuristics for the ATSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and its Variations, pp. 445–487. Kluwer Academic Publishers, Berlin (2002)
31. Jonker, R., Volgenant, A.: Assignment Solver Code. http://www.assignmentproblems.com/LAPJV.htm (1986)
32. Jonker, R., Volgenant, A.: Improving the Hungarian assignment algorithm. Oper. Res. Lett. **5**, 171–175 (1986)
33. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**, 325–340 (1987)
34. Kindervater, G., Volgenant, A., de Leve, G., van Gijlswijk, V.: On dual solutions of the linear assignment problem. Eur. J. Oper. Res. **19**(1), 76–81 (1985)
35. Kruskal, J.B.: On the shortest spanning tree of a graph and the traveling salesman problem. Proc. Am. Soc. **7**, 48–50 (1956)
36. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Res. Logist. Q. **2**, 83–97 (1955)
37. Lin, C.J., Wen, U.P.: Sensitivity analysis of the optimal assignment. Eur. J. Oper. Res. **149**(1), 35–46 (2003)
38. Malyshev, D.S., Pardalos, P.M.: Efficient computation of tolerances in the weighted independent set problem for some classes of graphs. Dokl. Math. **89**(2), 253–256 (2014)
39. Pettie, S.: Sensitivity analysis of minimum spanning trees in sub-inverse-ackermann time. In: Deng, X., Du, D., (eds.): ISAAC 2005, LNCS **3827** 964–973 (2005)
40. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM **49**(1), 16–34 (2002)
41. Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Technol. J. **36**, 1389–1401 (1957)
42. Ramaswamy, R., Orlin, J.B., Chakravarti, N.: Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs. Math. Program. Ser. A **102**, 355–369 (2005)
43. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**, 376–384 (1991)
44. Reinfeld, N.V., Vogel, W.R.: Mathematical Programming. Prentice-Hall, Englewood Cliffs, NJ (1958)
45. Richter, D., Goldengorin, B., Jager, G., Molitor, P.: Improving the efficiency of Helsgaun's Lin–Kernighan heuristic for the symmetric TSP. Lect. Notes Comput. Sci. **4852**, 99–111 (2007)

46. Salles da Cunha, A., Lucena, A.: Lower and upper bounds for the degree-constrained minimum spanning tree problem. Networks **50**(1), 55–66 (2007)
47. Shier, D.R., Witzgall, C.: Arc tolerances in minimum-path and network flow problems. Networks **10**, 277–291 (1980)
48. Tarjan, R.E.: Sensitivity analysis of minimum spanning trees and shortest path trees. Inf. Process. Lett. **14**(1), 30–33 (1982)
49. Toth, P.: Optimization engineering techniques for the exact solution of NP-hard combinatorial optimization problems. Eur. J. Oper. Res. **125**, 222–238 (2000)
50. Tsplib (Website to [43]). http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html
51. Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G.: Tolerance-based branch and bound algorithms for the ATSP. Eur. J. Oper. Res. **189**(3), 775–788 (2008)
52. Volgenant, A.: A Lagrangean approach to the degree-constrained minimum spanning tree problem. Eur. J. Oper. Res. **39**, 325–331 (1989)
53. Volgenant, A.: An addendum on sensitivity analysis of the optimal assignment. Eur. J. Oper. Res. **169**, 338–339 (2006)
54. Volgenant, T., Jonker, R.: The symmetric traveling salesman problem and edge exchanges in minimal 1-trees. Eur. J. Oper. Res. **12**, 394–403 (1983)