Algorithms for single-fault troubleshooting with dependent actions*

Václav Lín
Department of Decision-Making Theory
Institute of Information Theory and Automation
Czech Academy of Sciences
e-mail: lin@utia.cas.cz

Abstract

We study the problem of single-fault troubleshooting with dependent actions. We propose a binary integer programming formulation for the problem. This can be used to solve the problem directly or to compute lower bounds of optima using linear programming relaxation. We present an optimal dynamic programming algorithm, and three greedy algorithms for computing upper bounds of optima.

1 Introduction

We study *single-fault troubleshooting with dependent actions* [Heckerman et al., 1995, Jensen et al., 2001]. The problem is $\mathcal{NP}$-hard [Vomlelová and Vomlel, 2003], and it is a straightforward generalization of *min-sum set cover* and *pipelined-set cover* [Feige et al., 2004, Munagala et al., 2005]. These are combinatorial problems relevant in several areas other than automated repair.

We propose a binary integer programming formulation for the troubleshooting problem, and give several classes of additional valid inequalities. This can be used to solve the problem directly using a general purpose solver, or to compute lower bounds of optima by linear programming relaxation. We also describe several greedy algorithms for computing upper bounds of optima. We test the resulting lower and upper bounds in computational experiments.

Problem statement The troubleshooting problem studied in this paper may be stated as follows:

- A piece of equipment is faulty and the task is to construct a repair strategy with the least expected cost.

*This work was supported by the Czech Science Foundation through grant 13-20012S.

- There are m mutually exclusive and collectively exhaustive possible causes of the failure called *faults*. The faults are not directly observable. The equipment failure is caused by exactly one of the faults. Each fault F_i has nonzero probability of occurrence $P(F_i)$, and $\sum_{i=1}^{m} P(F_i) = 1$.
- There are n repair steps available, called *actions*, that can possibly remedy the failure. When performed, each action A_j can succeed or fail to fix the system failure, and it has a fixed cost $c(A_j)$ and a conditional probability of success $P(A_j \mid F_i) \geq 0$ for each fault F_i. In terms of probability, the actions are conditionally independent given the faults. It is assumed that an action that has failed once will fail again if performed. Hence, it is assumed that each action is performed at most once.
- The challenge is to find a suitable permutation of the actions $A_1, \ldots, A_n$, and use the permutation as a repair strategy: the actions are performed in the prescribed order until some of the actions succeeds (i.e. the equipment failure is repaired) or all the actions with nonzero probability of success have been used.

Let us denote by $\neg A$ the event that action A has failed and denote by

$$\mathbf{e}_j \quad = \quad \bigwedge_{k=1}^{j} \neg A_{\pi(k)} \ , \tag{1}$$

the information (called *evidence*) that the first j actions in permutation π have failed. Now, for a permutation of actions π we define

$$EC(\pi) \quad = \quad \sum_{i=1}^{n} c(A_{\pi(i)}) \cdot P(\mathbf{e}_{i-1}) \tag{2}$$

$$ECR(\pi) \quad = \quad EC(\pi) - \sum_{\substack{i=1,\ldots,n \\ P(A_{\pi(i)} \mid \mathbf{e}_{i-1})=0}} c(A_{\pi(i)}) \cdot P(\mathbf{e}_{i-1}) \ , \tag{3}$$

where EC is the *expected cost* of π, and ECR is the *expected cost of repair* of π, which is the expected cost of π where the actions with zero probability of success are skipped.

Our task is to find a permutation of actions minimizing the ECR. For some problems, a sequence minimizing EC minimizes also ECR. However, this is not a general rule as shown by Example 1.

Example 1. *We exhibit a troubleshooting problem where there are permutations π_1 and π_2 such that permutation π_1 minimizes ECR and $ECR(\pi_1) < ECR(\pi_2)$, permutation π_2 minimizes EC and $EC(\pi_1) > EC(\pi_2)$. In the problem we have two faults and two actions and the parameters are*

	A_1	A_2
$c(A)$	4	7

	F_1	F_2
$P(F)$	1/2	1/2

	F_1	F_2
$P(A_1 \mid F)$	1	0
$P(A_2 \mid F)$	1	1/2

Let $\pi_1 = \langle A_2, A_1 \rangle$ *and* $\pi_2 = \langle A_1, A_2 \rangle$. *The expected costs are:*

	π_2	π_1
EC	**7.5**	8
ECR	7.5	**7**

Special cases When the outcomes of actions are deterministic, i.e. the probability $P(A \text{ succeeds } \mid F \text{ is present})$ is either zero or one for all combinations of action A and fault F, then the problem reduces to the *pipelined set cover* problem Munagala et al. [2005]. When it is further assumed that all the action cost and fault probabilities are uniform, the problem is equivalent to the *min-sum set cover* problem Feige et al. [2004].

Contribution and structure of the paper The main contribution of the paper is a binary integer programming formulation for single fault troubleshooting with dependent actions. The formulation is useful in two ways:

1. The formulation can be used for solving the troubleshooting problem directly with any general purpose integer programming solver.
2. Even if the integer program at hand is too hard to solve to optimality, we may nonetheless use it to compute lower bounds of optima by linear programming relaxation. These bounds may be used in special purpose branch & bound algorithms for troubleshooting (such as the algorithm given by Vomlelová and Vomlel [2003]).

The binary integer programming formulation is described in Section 5. In Section 4 we describe three simple greedy algorithms for solving the problem. These algorithms are useful because the search for an optimal permutation of actions by branch & bound algorithms is often greatly facilitated by having a good upper bound of the optimum. The paper concludes with brief discussion of computational experience in Section 6.

2 Notation

The set of all faults is $\mathcal{F} = \{F_1, \ldots, F_m\}$, the set of all actions is $\mathcal{A} = \{A_1, \ldots, A_n\}$. For an action A, the set of all faults that can be repaired by action A is $\mathcal{F}(A)$; similarly, $\mathcal{A}(F)$ is the set of actions that may repair fault F:

$$\begin{aligned} \mathcal{F}(A) &= \{F \in \mathcal{F} \colon P(A \mid F) > 0\} \ , \\ \mathcal{A}(F) &= \{A \in \mathcal{A} \colon P(A \mid F) > 0\} \ . \end{aligned}$$

Let π be a permutation of the actions. With the notation just introduced and using the assumptions of mutually exclusive faults and conditional independence of actions given faults, the expected cost may be written as

$$EC(\pi) = \sum_{A \in \mathcal{A}} c(A) \cdot \sum_{F \in \mathcal{F}} P(F) \cdot \prod_{\substack{B \in \mathcal{A}(F) \\ \pi(B) < \pi(A)}} P(\neg B | F) \ . \tag{4}$$

When we perform an action A and the action fails, we nevertheless obtain some information. In particular, the marginal probability distribution $P(\mathcal{F})$ changes to $P(\mathcal{F} \mid \neg A)$. As mentioned above, we call this information *evidence*. For consistency, we define $\mathbf{e}_0$, the void *initial evidence* that we have before any of the actions has been executed. We define

$$\mathcal{A}(\mathbf{e}) \;=\; \{A \in \mathcal{A}\colon P(A \mid \mathbf{e}) > 0\}\ .$$

It is assumed that once failed action will fail again if performed. In terms of probability, $P(A \mid \neg A)$ is zero, and hence the set $\mathcal{A}(\mathbf{e})$ does not contain any of the actions that are included in $\mathbf{e}$.

3 Dynamic programming

A dynamic programming approach to troubleshooting was first proposed by Vomlelová and Vomlel [2003]. The problem studied in this paper can be solved by dynamic programming using recurrence

$$ECR^\star(\mathbf{e}) \quad = \quad \min_{A \in \mathcal{A}(\mathbf{e})} \big[c(A \mid \mathbf{e}) + P(\neg A \mid \mathbf{e}) \cdot ECR^\star(\mathbf{e} \wedge \neg A)\big]\ , \qquad (5)$$

where the minimum on the right hand side equals zero if taken over an empty set. Now, $ECR^\star(\mathbf{e}_0)$ is the optimal expected cost of the troubleshooting problem.

4 Greedy algorithms

Greedy polynomial-time algorithms may be used to construct permutations of actions that are not guaranteed to be optimal but experience shows that very often they are optimal or "nearly optimal". We shall describe three such algorithms in this section.

Algorithm Updating P/C Perhaps the most natural greedy algorithm is one called Updating P/C [Jensen et al., 2001] At i^{th} step, $i = 1, \ldots, n$, the algorithm selects an action $A \in \mathcal{A}(\mathbf{e}_{i-1})$ maximizing the ratio

$$\frac{P(A \mid \mathbf{e}_{i-1})}{c(A)}\ .$$

For minimization of EC, Kaplan et al. [2005] proved that Updating P/C has a guaranteed approximation factor: it never returns a sequence with expected cost greater than four times the optimum. By the complexity-theoretic results of Feige et al. [2004], that is most likely the best guaranteed approximation factor possible.

Algorithm DP-greedy Another greedy algorithm is motivated by the dynamic programming recurrence (5). We shall call the algorithm DP-GREEDY. At i^{th} step, the algorithm selects an action $A \in \mathcal{A}(\mathbf{e}_{i-1})$ minimizing

$$c(A) + P(\neg A \mid \mathbf{e}_{i-1}) \cdot \widetilde{EC}(\mathbf{e}_{i-1} \wedge \neg A) \ ,$$

where $\widetilde{EC}(\mathbf{e})$ denotes an estimate of the expected cost of optimal sequence of the remaining actions from $\mathcal{A}(\mathbf{e})$. The estimate is computed by the UPDATING P/C algorithm. Although seemingly different, algorithm DP-GREEDY is equivalent to a greedy algorithm proposed by Langseth and Jensen [2001].[1]

Algorithm I-greedy The last greedy algorithm uses an information-theoretic criterion for selection of the hopefully best action given evidence $\mathbf{e}$. We call it I-GREEDY. It is inspired by the ID3 algorithm [Quinlan, 1986]. Let $H(\mathcal{F} \mid \mathbf{e})$ be the Shannon entropy of marginal distribution $P(\mathcal{F} \mid \mathbf{e})$, that is

$$H(\mathcal{F} \mid \mathbf{e}) \quad = \quad -\sum_{F \in \mathcal{F}} P(F \mid \mathbf{e}) \cdot \log P(F \mid \mathbf{e}) \ ,$$

and let $I(A \mid \mathbf{e})$ be the *information gain* of performing action A, i.e. the expected decrease of $H(\mathcal{F} \mid \mathbf{e})$ induced by performing A:

$$I(A \mid \mathbf{e}) \quad = \quad H(\mathcal{F} \mid \mathbf{e}) - \Big[P(A \mid \mathbf{e}) \cdot H(\mathcal{F} \mid \mathbf{e} \wedge A) + P(\neg A \mid \mathbf{e}) \cdot H(\mathcal{F} \mid \mathbf{e} \wedge \neg A)\Big].$$

Given evidence $\mathbf{e}$, it seems desirable to select an action maximizing ${}^{I(A \mid \mathbf{e})}/_{c(A)}$. However, we do not want the selection criterion to be biased towards actions with high entropy $H(\mathcal{F} \mid \mathbf{e} \wedge A)$ since we are not interested in the entropy of $P(\mathcal{F})$ in the case that A suceeds. To this end, we assume $H(\mathcal{F} \mid \mathbf{e} \wedge A)$ to be zero for all A and $\mathbf{e}$, and we select at each step an action $A \in \mathcal{A}(\mathbf{e}_{i-1})$ maximizing

$$\frac{H(\mathcal{F} \mid \mathbf{e}) - P(\neg A \mid \mathbf{e}_{i-1}) \cdot H(\mathcal{F} \mid \mathbf{e}_{i-1} \wedge \neg A)}{c(A)} \ . \tag{6}$$

5 Integer linear programming formulation

We shall formulate an integer linear program encoding the troubleshooting problem. For background information about integer programming we refer to [Wolsey, 1998]. For linear programming, the classic reference is [Dantzig, 1998].[2]

To encode a permutation of actions from the set $\mathcal{A}$, we use binary variables $d_{A,B}$ for every pair of distinct actions $A, B \in \mathcal{A}$. Given a permutation π of the

[1] The proof is to be found in [Lín, 2015].

[2] The *(binary) integer programming* problem is this: given an r-element vector $\mathbf{c}$, an s-element vector $\mathbf{b}$ and an $(r \times s)$-matrix $\mathbf{A}$, find an r-element vector $\mathbf{x}$ minimizing $\mathbf{c}^T\mathbf{x}$ subject to constraints $\mathbf{x} \in X \cap \{0,1\}^r$ where $X = \{\mathbf{x} \in \Re^r \colon \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$. The set X is a *formulation* for the set of feasible solutions. Lower bound of the optimum can be obtained by a *linear programming relaxation*: we solve the linear programming problem $\mathbf{c}^T\mathbf{x} \to \min$ subject to constraints $\mathbf{x} \in X$, $\mathbf{x} \geq \mathbf{0}$. The linear programming relaxation can be computed in polynomial time, whereas binary integer programming is $\mathcal{NP}$-hard.

actions, we have $d_{A,B} = 1$ if action A precedes action B in the permutation π, otherwise $d_{A,B} = 0$. Variables $d_{A,B}$ should encode a linear ordering relation on $\mathcal{A}$. That means that the relation is *asymmetric* and *transitive* To enforce the requirement of asymmetry, we introduce equation (7) for each ordered pair of distinct actions A, B. To enforce the requirement of transitivity, we add inequality (8) for every ordered triple of pairwise distict actions A, B, C.

$$d_{A,B} = 1 - d_{B,A} \ . \tag{7}$$

$$d_{A,B} + d_{B,C} \leq d_{A,C} + 1 \ . \tag{8}$$

We now proceed to formulate the expected cost of action sequence as a linear function. For simplicity, we begin by EC and turn to ECR later. Assuming that a fixed permutation π is encoded by variables $d_{A,B}$ introduced above, we can write (4) as:

$$EC = \sum_{A \in \mathcal{A}} c(A) \cdot \sum_{F \in \mathcal{F}} P(F) \cdot \prod_{\substack{B \in \mathcal{A}(F) \setminus \{A\} \\ d_{B,A}=1}} P(\neg B \mid F) \ . \tag{9}$$

(Whenever the product in (9) is taken over an empty set of factors, we assume that the product equals one. That is $\prod_{B \in \emptyset} P(\neg B \mid F) = 1$.) Minimizing (4) is equivalent to minimizing (9) subject to the constraints (7) and (8). To express (9) as a linear function, we introduce a binary variable $x_{F,A,\mathcal{B}}$ for each fixed combination of fault F, action A and a set of actions $\mathcal{B} \subseteq \mathcal{A}(F) \setminus \{A\}$. The value of $x_{F,A,\mathcal{B}}$ is defined as

$$x_{F,A,\mathcal{B}} = \Big(\prod_{B \in \mathcal{B}} d_{B,A} \Big) \cdot \Big(\prod_{\substack{B \in \mathcal{A}(F) \setminus \mathcal{B} \\ B \neq A}} d_{A,B} \Big) \ . \tag{10}$$

In words, variable $x_{F,A,\mathcal{B}}$ equals one if and only if all the actions $B \in \mathcal{B}$ precede action A, and all the remaining actions from $\mathcal{A}(F)$ are preceded by A. Associated to each variable $x_{F,A,\mathcal{B}}$ is a coefficient

$$Q_{F,A,\mathcal{B}} = c(A) \cdot P(F) \cdot \prod_{B \in \mathcal{B}} P(\neg B \mid F) \ .$$

For $\mathcal{B} = \emptyset$, we have $Q_{F,A,\mathcal{B}} = c(A) \cdot P(F)$. For any fixed fault F and action A, exactly one of the variables $x_{F,A,\mathcal{B}}$ equals one. With this observation, we may replace the nonlinear objective (9) by a linear function

$$EC = \sum_{A \in \mathcal{A}} \sum_{F \in \mathcal{F}} \sum_{\substack{\mathcal{B} \subseteq \mathcal{A}(F) \\ \mathcal{B} \not\ni A}} Q_{F,A,\mathcal{B}} \cdot x_{F,A,\mathcal{B}} \ . \tag{11}$$

The number of summands in (11) is exponential in the size of sets $\mathcal{A}(F)$. However, we can assume that in practical applications, the size $|\mathcal{A}(F)|$ is bounded from above by a reasonably small constant.

To express definition (10) in terms of linear inequalities, we observe that the definition implies for each fixed combination $F, A, \mathcal{B}$:

$$x_{F,A,\mathcal{B}} \quad \geq \quad 1 + \sum_{B \in \mathcal{B}} (d_{B,A} - 1) + \sum_{\substack{B \in \mathcal{A}(F) \setminus \mathcal{B} \\ B \neq A}} (d_{A,B} - 1). \tag{12}$$

Bounding the variables $x_{F,A,\mathcal{B}}$ from above is not necessary since all the coefficients in (11) are nonegative. In case that $\mathcal{A}(F) \setminus \{A\}$ is an empty set, we have $x_{F,A,\emptyset} = 1$. To summarize, for minimization of EC we have a minimization linear program with objective function (11) and constraints (12), (7), (8).

To minimize ECR rather than EC, we need to add to the formulation additional variables and constraints. We say that an action A_i is *dominated* in permutation π if its success probability $P(A_{\pi(i)} \mid \mathbf{e}_{i-1})$ is zero. In the linear model, we may define additional "correction" variable $w_{F,A,\mathcal{B}}$ with coefficient '$-Q_{F,A,\mathcal{B}}$' for every variable $x_{F,A,\mathcal{B}}$. Logically, the variable is defined

$$w_{F,A,\mathcal{B}} \Longleftrightarrow x_{F,A,\mathcal{B}} \wedge \Big[\bigwedge_{G \in \mathcal{F}(A)} \bigvee_{\substack{B \in \mathcal{A}(G) \\ B \neq A \\ P(B|G)=1}} d_{BA} \Big]$$

In words, variable $w_{F,A,\mathcal{B}}$ equals one if and only if variable $x_{F,A,\mathcal{B}}$ equals one and action A is dominated. The linear objective function is then

$$ECR \;=\; \sum_{x_{F,A,\mathcal{B}}} Q_{F,A,\mathcal{B}} \cdot x_{F,A,\mathcal{B}} - \sum_{w_{F,A,\mathcal{B}}} Q_{F,A,\mathcal{B}} \cdot w_{F,A,\mathcal{B}} \tag{13}$$

where the sums extend over all the x- and w-variables. To express the definition of w by linear inequalities, we observe that only upper bound for the w-variables is needed (since their coefficients are negative), and it is sufficient to introduce linear constraints

$$w_{F,A,\mathcal{B}} \quad \leq \quad x_{F,A,\mathcal{B}} \tag{14}$$

$$(\forall G \in \mathcal{F}(A)) \quad w_{F,A,\mathcal{B}} \quad \leq \quad \sum_{\substack{B \in \mathcal{A}(G) \\ B \neq A \\ P(B|G)=1}} d_{B,A} \tag{15}$$

To keep the linear relaxation tight, we *do not* add to the model variables w that either cannot ever equal one or have zero coefficient Q. To be included in the linear programming formulation, the following conditions must be satisfied for variable $w_{F,A,\mathcal{B}}$:

1. Action A can be dominated. That means that for every fault $G \in \mathcal{F}(A)$ there is an action $B \in \mathcal{A}(G) \setminus \{A\}$ that can solve fault G perfectly, i.e. $P(B \mid G) = 1$.

2. The coefficient $Q_{F,A,\mathcal{B}}$ is nonzero. That means that for every action $B \in \mathcal{B}$, the probability $P(\neg B \mid F)$ is nonzero.

Moreover, we do not add to the model any variable $w_{F,A,\mathcal{B}}$ such that the action A can solve fault F (in our notation $A \in \mathcal{A}(F)$). Indeed, consider a variable $w_{F,A,\mathcal{B}}$ with action $A \in \mathcal{A}(F)$. If it has nonzero coefficient $Q_{F,A,\mathcal{B}}$, then it means that action A is not dominated, as it is not preceded by any action B with $P(B \mid F) = 1$.

5.1 Classes of additional valid inequalities

Linear programming relaxation is obtained from the integer program by replacing the integrality requirement $d_{A,B} \in \{0, 1\}$ by $0 \leq d_{A,B} \leq 1$ for all the d-variables and likewise for all the x- and w-variables. In general, objective value of linear programming relaxation is a lower bound of the objective value of the minimization integer program. To make the bound as tight as possible, we may add to the linear model additional *valid inequalities*. That is, inequalities that are satisfied by all feasible integer solutions.

The first class of valid inequalities is based on the observation that for any fixed combination of a fault F and an action A, exactly one of the variables $x_{F,A,\mathcal{B}}$ equals one, i.e.

$$\sum_{\substack{\mathcal{B} \subseteq \mathcal{A}(F) \\ \mathcal{B} \not\ni A}} x_{F,A,\mathcal{B}} \quad = \quad 1 \ . \tag{16}$$

Another class of valid inequalities is based on observing that given fault F and a fixed permutation of actions, there is exactly one action $A \in \mathcal{A}(F)$ that is not preceded by any other action $B \in \mathcal{A}(F)$. That is, for every fault F we have:

$$\sum_{A \in \mathcal{A}(F)} x_{F,A,\emptyset} \quad = \quad 1 \ . \tag{17}$$

We observe that if action B precedes action A, and action A precedes all the actions from $\mathcal{A}(F)$, then also action B precedes all the actions from $\mathcal{A}(F)$. Hence, for any fixed combination of fault F and distinct actions A and B we have:

$$x_{F,A,\emptyset} + d_{B,A} \quad \leq \quad x_{F,B,\emptyset} + 1 \ . \tag{18}$$

Another idea for valid inequalities is based on the fact that if an action is dominated in optimal sequence, then so should be its successors. Therefore, for all distinct actions A and B neither of which belongs to $\mathcal{A}(F)$, we have

$$w_{F,A,\emptyset} + d_{A,B} \quad \leq \quad w_{F,B,\emptyset} + 1 \ . \tag{19}$$

For any fixed triple $F, A, \mathcal{B}$, a combination of (10) and (14) yields inequalities

$$w_{F,A,\mathcal{B}} \quad \leq \quad d_{B,A} \text{ for every } B \in \mathcal{B} \tag{20}$$

$$w_{F,A,\mathcal{B}} \quad \leq \quad d_{A,B} \text{ for every } B \in \mathcal{A}(F) \setminus \mathcal{B} \setminus \{A\} \ . \tag{21}$$

Another class of valid inequalities that we devise is inspired by a heuristic function due to Vomlelová and Vomlel [2003]. For any given fault F we can find a permutation π_F of all the actions minimizing

$$z(\pi) \quad = \quad \sum_{A \in \mathcal{A}} c(A) \cdot \prod_{\substack{B \in \mathcal{A} \\ \pi(B) < \pi(A)}} P(\neg B \mid F) \ .$$

The minimizing permutation π_F is found by ordering the actions in $\mathcal{A}$ so that the ratios $P(A \mid F)/c_A$ are nonincreasing. With this observation, we can construct constraints (22) for each fault F:

$$\sum_{A \in \mathcal{A}} \sum_{\substack{\mathcal{B} \subseteq \mathcal{A}(F) \\ \mathcal{B} \not\ni A}} Q_{F,A,\mathcal{B}} \cdot x_{F,A,\mathcal{B}} \quad \geq \quad P(F) \cdot z(\pi_F) \ . \tag{22}$$

The heuristic function of Vomlelová and Vomlel [2003] can be expressed by a single inequality:

$$\sum_{x_{F,A,\mathcal{B}}} Q_{F,A,\mathcal{B}} \cdot x_{F,A,\mathcal{B}} - \sum_{w_{F,A,\mathcal{B}}} Q_{F,A,\mathcal{B}} \cdot w_{F,A,\mathcal{B}} \quad \geq \quad \sum_{F \in \mathcal{F}} P(F) \cdot z(\pi_F) \ . \tag{23}$$

The sums in (23) are taken over all the x- and w-variables that exist in the linear programming formulation.

Fixing partial order of actions in advance In some cases, we can fix a partial order of some of the actions before starting to search for an optimal sequence, thereby reducing the number of sequences that need to be considered. In particular, we may use the following proposition.[3]

Proposition 1. *Let* $\mathbf{s}$ *be an optimal sequence of actions. Let there be two distinct actions* A *and* B *in* $\mathbf{s}$ *such that:*

- *the sets* $\mathcal{F}(A)$ *and* $\mathcal{F}(B)$ *are disjoint,*
- *there is no action* C *with set* $\mathcal{F}(C)$ *insersecting* $\mathcal{F}(A) \cup \mathcal{F}(B)$.

Further, assume that action A *precedes action* B *in sequence* $\mathbf{s}$ *(the two actions are not necessarily adjacent). Then* $P(A)/c(A) \geq P(B)/c(B)$.

We may use Proposition 1 to construct a partial ordering of actions satisfying the conditions stated in the proposition. Once such an ordering is constructed, we may fix the corresponding precedence variables $d_{A,B}$ to appropriate values.

[3] Proposition 1 is a straightforward generalization of a theorem proved by Jensen et al. [2001]. The proof is to be found in [Lín, 2015].

Cutting planes procedure The basic integer programming formulation contains inequalities (12),(7),(8), (14) and (15). The formulation may be strenghtened by adding additional valid equalities and inequalities mentioned above. However for computational reasons, we do not add them all at once, but rather in an iterative fashion. The additional constraints are conventionally called *cutting planes*. The procedure of adding cutting planes can be outlined as follows:

1. Construct the initial integer programming formulation and compute its relaxation by replacing the integrality requirements $v \in \{0, 1\}$ by $0 \leq v \leq 1$ for every variable v of the formulation.
   Let X denote the obtained linear programming formulation, and let $\mathbf{x}$ denote its solution vector found by linear programming.

2. For each class of valid inequalities or equalities[4] listed in Section 5.1:

   (a) Investigate whether some inequalities of the class are violated by the current solution vector $\mathbf{x}$.

   (b) Add the violated inequalities to X and solve by linear programming.

   (c) If the addition of violated inequalities in the previous step lead to increase in the objective value, keep the inequalities in X. Otherwise, remove them.

   (d) Remove from the formulation cutting planes that are not satisfied with equality (i.e they have nonzero "slack value").

3. Repeat the previous step until the objective value does not increase, or the number of iterations exceeds some predetermined parameter, or $\mathbf{x}$ is an integral vector.

A more detailed description of the procedure is to appear in [Lín, 2015].

6 Computational experience

In this last section we collect results of a small computational study. The algorithms described in the paper were run on nine problems. One of the problems was generated, the other were extracted from real world troubleshooting models. Full details of the models cannot be given for confidentiality reasons, so we provide only some basic characteristics in Table 1. More details can be provided upon request.

We investigate tightness of the upper bounds computed by greedy algorithms DP-GREEDY, UPDATING P/C and I-GREEDY. The tightness is measured by ratio of the upper bound to the optimal ECR. The optima are computed by dynamic programming and/or by integer programming. The results are in Table 2. In the same table are ratios of lower bounds to optimal ECR. The lower bounds are computed by the heuristic function of Vomlelová and Vomlel [2003]

[4] In the following, we say just "inequalities" instead of "inequalities or equalities".

		$c(A)$			$P(F)$		$P(A \mid F) \neq 0$		
	$\lvert\mathcal{A}\rvert$	μ	σ	$\lvert\mathcal{F}\rvert$	μ	σ	%	μ	σ
1	25	6,840	3,923	26	0,038	0,037	8,800	0,886	0,125
2	13	24,231	30,868	12	0,083	0,055	9,600	0,941	0,066
3	7	10,429	11,588	6	0,167	0,158	23,800	0,898	0,175
4	13	28,154	31,945	13	0,077	0,093	15,380	0,950	0,050
5	10	1,000	0,000	10	0,100	0,000	27,000	0,929	0,051
6	14	83,000	264,406	13	0,077	0,056	24,720	0,931	0,092
7	20	10,900	7,840	26	0,039	0,033	6,920	0,930	0,200
8	13	34,690	40,400	12	0,083	0,078	20,510	0,963	0,092
9	11	26,450	35,708	11	0,091	0,118	15,700	0,935	0,068

Table 1: For each problem, we give the number of actions $|\mathcal{A}|$ and number of faults $|\mathcal{F}|$. We give mean μ and standard deviation σ for action costs $c(A)$ and fault probabilities $P(F)$. For probability distribution $P(\mathcal{A} \mid \mathcal{F})$ we give the percentage (%), mean and standard deviation of nonzero entries.

	DP-G.	UP. P/C	I-G.	heur.	LP	LP+cuts	cuts used
1	1,000	1,000	1,003	0,590	0,470	0,687	(22)
2	1,005	1,006	1,018	0,563	0,973	0,973	—
3	1,000	1,000	1,000	0,867	0,796	0,867	(23)
4	1,000	1,000	1,024	0,619	0,742	0,861	(16), (17), (18)
5	1,000	1,047	1,047	0,436	0,379	0,562	(22)
6	1,000	1,000	1,022	0,856	0,883	0,936	(17), (18)
7	1,001	1,010	1,014	0,648	0,913	0,926	(18)
8	1,000	1,000	1,000	0,630	0,417	0,750	(17), (18)
9	1,000	1,000	1,019	0,699	0,861	0,892	(22)

Table 2: Tightness of bounds of ECR. On the left are ratios of upper bounds to optimal ECR. On the right are ratios of lower bounds to optimal ECR.

	heur.	LP	LP+cuts
1	0,577	0,564	0,822
2	0,563	0,973	0,973
3	0,818	0,787	0,942
4	0,619	0,742	0,861
5	0,436	0,379	0,562
6	0,856	0,918	0,998
7	0,609	0,983	0,983
8	0,619	0,539	0,827
9	0,699	0,861	0,892

Table 3: Tightness of lower bounds of EC.

(column "heur."), by linear programming relaxation without cutting planes (column "LP") and by linear programming relaxation with cutting planes involved (column "LP+cuts"). The rightmost column indicates which inequalities were used as cutting planes. In Table 3 we give similar results for the lower bounds when EC is optimized rather than ECR. We see that the greedy algorithms provide solutions that are always either optimal or very close to optimal. The algorithm DP-GREEDY performs very well and finds an optimal solution in most cases[5]. As far as the lower bounds are concerned, adding cutting planes to the basic linear programming formulation leads to a tighter bound in all cases but one. In general, lower bounds computed by the cutting planes procedure are the strongest. In one case however, they are no better than the bounds provided by the simple heuristic proposed by Vomlelová and Vomlel [2003]. We also note that the linear programming relaxation provides tighter bounds when optimizing EC rather than ECR. This is natural, since the w-variables in (13) generally decrease the value of the relaxation.

Acknowledgement

I thank Jiří Vomlel and Thorsten Ottosen for their help. I would like to acknowledge using the PuLP library for Python and CLP/CBS solvers. All these tools are part of the COIN-OR project [Lougee-Heimer, 2003].

References

George B. Dantzig. *Linear programming and extensions.* Princeton University Press, 1998.

Uriel Feige, László Lovász, and Prasad Tetali. Approximating Min Sum Set Cover. *Algorithmica*, 40(4):219–234, 2004.

David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995.

Finn Verner Jensen, Uffe Kjærulff, Brian Kristiansen, Helge Langseth, Claus Skaanning, Jiří Vomlel, and Marta Vomlelová. The SACSO methodology for troubleshooting complex systems. *AI EDAM*, 15(4):321–333, 2001.

Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 356–365. ACM, 2005.

Helge Langseth and Finn Verner Jensen. Heuristics for two extensions of basic troubleshooting. In *IN: Seventh Scandinavian conference on Artificial Intelligence, SCAI'01, Frontiers in Artificial Intelligence and applications, IOS*, 2001.

[5]Without providing a proof that the solution is in fact optimal.

Václav Lín. Forthcoming thesis, 2015.

Robin Lougee-Heimer. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In Thomas Eiter and Leonid Libkin, editors, *ICDT*, volume 3363 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2005. ISBN 3-540-24288-0.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Marta Vomlelová and Jiří Vomlel. Troubleshooting: NP-hardness and solution methods. *Soft Computing*, 7(5):357–368, 2003.

Laurence A. Wolsey. *Integer programming*. Wiley, New York, 1998.