Extensions of Decision-Theoretic Troubleshooting: Cost Clusters and Precedence Constraints

Václav Lín*

Institute of Information Theory and Automation of the ASCR,
Pod Vodárenskou věží 4, CZ-182 08, Prague, Czech Republic
lin@utia.cas.cz

Abstract. In decision-theoretic troubleshooting [5,2], we try to find a cost efficient repair strategy for a malfunctioning device described by a formal model. The need to schedule repair actions under uncertainty has required the researchers to use an appropriate knowledge representation formalism, often a probabilistic one.

The troubleshooting problem has received considerable attention over the past two decades. Efficient solution algorithms have been found for some variants of the problem, whereas other variants have been proven *NP*-hard [5,2,4,17,16].

We show that two troubleshooting scenarios — *Troubleshooting with Postponed System Test* [9] and *Troubleshooting with Cost Clusters without Inside Information* [7] — are *NP*-hard. Also, we define a troubleshooting scenario with precedence restrictions on the repair actions. We show that it is *NP*-hard in general, but polynomial under some restrictions placed on the structure of the precedence relation. In the proofs, we use results originally achieved in the field of Scheduling. Such a connection has not been made in the Troubleshooting literature so far.

Keywords: Computational Complexity, Dynamic Programming, Decision-Theoretic Troubleshooting, Scheduling.

1 Introduction

Suppose a man-made device failed to work – the exact cause of the failure is unknown, and the possible steps to fix it are costly and not 100% reliable. Any attempt to resolve the problem may fail, but the incurred cost has to be paid in any case. Solving problems such as this one has lead to development of the field of decision-theoretic troubleshooting [5,2]. The need to decide under uncertainty has necessitated the use of an appropriate formalism. Bayesian networks have been adopted for their clear semantics and computational tractability. The troubleshooting problem is known to be polynomial in few special cases and *NP*-hard

* This work was supported by the Ministry of Education of the Czech Republic through grant 1M0572 and by the Czech Science Foundation through grant ICC/08/E010.

W. Liu (Ed.): ECSQARU 2011, LNAI 6717, pp. 206–216, 2011.

in others [5,2,4,17,16]. Decision-theoretic troubleshooting has been successfully applied in the area of printer diagnosis and maintenance [2,13].

In this paper, we build on earlier work published in [4,7,9] and provide new results on computational complexity of the problems studied in the cited papers. In the first part – Section 2 – we give an overview of the troubleshooting scenarios studied in earlier literature. Specifically, Section 2.1 describes a very basic troubleshooting scenario taken from [5,2,4]. In the same Section, we define a novel scenario by adding a precedence relation on the troubleshooting actions. In Section 2.2, we review a generalization of basic troubleshooting — *Troubleshooting with Postponed System Test* – studied recently in [9], and describe a novel Dynamic Programming solution. In Section 2.3 we introduce a more general scenario of *Troubleshooting with Cost Clusters without Inside Information*, originally defined in paper [7].

The main results of the paper are found in Section 3. In Section 3.2, we show that troubleshooting with a postponed system test is *NP*-hard. As a corollary, troubleshooting with cost clusters without inside information is shown to be *NP*-hard in the same Section. In Section 3.3, we turn to troubleshooting with precedence constraints and show that it is *NP*-hard, but solvable in polynomial time when the precedence relation has a structure of series parallel directed graph [6,15]. In the *NP*-hardness proofs, we use results achieved originally in the field of Scheduling. We consider this to be one of the contributions of the paper, since such a connection has not been made in the Troubleshooting literature so far. In Section 4, we sum up the results and conclude the paper by suggesting directions for future research.

2 Troubleshooting Scenarios

Before proceeding with the formal definitions, we will illustrate the troubleshooting scenarios on a simple example inspired by [7,17].

Imagine you are printing a report but the colors come out very light. You have several options to choose from: restart the printer, change the print settings, reseat the toner cartridge or get a new cartridge altogether. These actions differ both in their difficulty and in the likelihood of fixing the print problem. You need to decide how to sequence the available repair actions so that the "expected difficulty level" of the repair is as low as possible. This kind of problem is solved in the basic troubleshooting scenario, described in Section 2.1.

Continuing with our print example, imagine that the printing itself is very expensive – you have to think twice before performing a test print to check that the repair actions have actually helped. Troubleshooting problems such as this one are defined in Section 2.2.

To make the example yet more complicated, assume that you are troubleshooting a complex industrial printer and some of the repair actions are only available after disassembling parts of the machine – and different actions may require different parts to be disassembled. To perform a test print, the machine has to be reassembled. Section 2.3 covers problems of this kind.

2.1 Basic Troubleshooting

The *Basic Troubleshooting* problem is given by

- a set $\mathcal{F} = \{F_1, \ldots, F_m\}$ of possible faults,
- a set $\mathcal{A} = \{A_1, \ldots, A_n\}$, $n \geq m$, of available repair steps,
- a probabilistic model $P(\mathcal{F} \cup \mathcal{A})$ describing interactions between the elements of $\mathcal{A}$ and $\mathcal{F}$.

Each action A_i bears a constant cost $c(A_i)$ and can either fail ($A_i = 0$) or fix the fault ($A_i = 1$). The following assumptions are made:

- There is exactly one fault present in the modeled system.
- Each action addresses exactly one fault.
- The model $P(\mathcal{F} \cup \mathcal{A})$ satisfies conditional independence assumptions encoded by the Bayesian network shown in Figure 1. Specifically, the actions are conditionally independent given the faults.

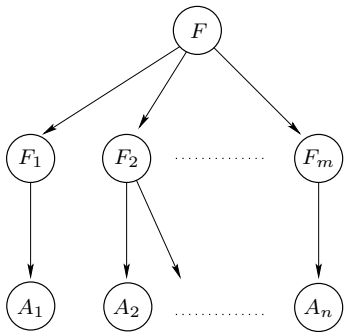

Fig. 1. Bayesian network encoding conditional independence of actions given faults. F is the fault variable with possible values $f_1, \ldots, f_m$. Variables $F_1, \ldots, F_m$ are Boolean indicators with $P(F_i = 1 | F = f_i) = 1$. Variables $A_1, \ldots, A_n$ are also Boolean. Each A_j has exactly one parent in the graph.

We assume that no new faults are introduced during the troubleshooting session, and the result of any action stays the same during a session. These two last assumptions are only implicit in most of the papers on troubleshooting. They are explicitly stated only in more recent papers, such as [9].

Let $\pi = \{\pi(1), \ldots, \pi(n)\}$ denote a permutation of indices $1, \ldots, n$; then the *troubleshooting strategy* is a sequence $A_{\pi(1)}, \ldots, A_{\pi(n)}$ of actions performed until the fault is fixed or all actions are exhausted. Thus, the action $A_{\pi(i)}$ will be performed only if all the preceding actions fail, that is, $A_j = 0$ for $j = \pi(1), \ldots, \pi(i-1)$. To solve the troubleshooting problem, we have to find a repair strategy with the lowest *Expected Cost of Repair*:

$$ECR(A_{\pi(1)}, \cdots, A_{\pi(n)}) = \sum_{i=1}^{n} c(A_{\pi(i)}) \cdot P\Big(\bigwedge_{j<i} \{A_{\pi(j)} = 0\}\Big)$$

The following proposition describes an easy method of finding the optimal troubleshooting sequence and computing its ECR.

Proposition 1 (Jensen et al., 2001 [4]). *Under the assumptions of*

- *single fault,*
- *each action addressing exactly one fault,*
- *conditional independence of actions given the faults,*

the optimal troubleshooting sequence is found in $O(n \cdot \log n)$ *time by ordering the actions so that ratio values* $P(A_j = 1)/c(A_j)$ *are decreasing where*

$$P(A_j = 1) = \sum_i P(A_j = 1 | F = f_i) \cdot P(F = f_i).$$

Further, the ECR can be computed as

$$ECR(A_{\pi(1)}, \cdots, A_{\pi(n)}) = \sum_{i=1}^{n} c(A_{\pi(i)}) \cdot \big(1 - \sum_{j<i} P(A_{\pi(j)} = 1)\big).$$

Proof. See Proposition 1, Proposition 2 and Theorem 1 in [4]. The $O(n \cdot \log n)$ time requirement is given by the complexity of sorting (see, e.g., [11]). □

The assumptions of Proposition 1 are quite restrictive; however, weakening them often yields an *NP*-hard scenario as shown in papers [17,16].

The single fault assumption is relaxed in paper [14], where *multiple faults* are considered. However, it is assumed that the faults as well as actions are independent. The optimal sequence is found by ordering the actions so that the ratio values

$$\frac{P(A_j = 1)}{c(A_j) \cdot (1 - P(A_j = 1))}$$

are decreasing.

Troubleshooting with Precedence Constraints. In some applications, there may be restrictions imposed on the order of troubleshooting actions – some of the actions become available only after performing some other actions. A restriction on the order of actions does not typically correspond to the probabilistic dependence of actions. For an example of such an application, see Section 4.1 in paper [4].

Formally, we assume that there is a precedence relation described by an acyclic directed graph G with vertices labeled by the actions from $\mathcal{A}$. A troubleshooting sequence $A_{\pi(1)}, \cdots, A_{\pi(n)}$ is valid only if $A_{\pi(j)}$ is not a predecessor of $A_{\pi(i)}$ in G for all $i < j$. We will show in Section 3.3 that *Basic Troubleshooting with Precedence Constraints* is *NP*-hard for a general acyclic directed graph G, but is solvable in polynomial time for a wide class of *series parallel* graphs.

2.2 Troubleshooting with Postponed System Test

Following [9], we add the assumption that, after performing a troubleshooting action, we do not know whether the action has solved the problem. We have to perform a system test, requiring additional cost c_D, to find out whether the problem has been fixed. The need to schedule system tests complicates construction of the optimal sequence – when the system test is postponed too much, we risk performing needless repair actions; when we perform the test too early to check the system state, we risk missing a necessary repair action.

To solve the troubleshooting problem, we construct an *ordered partition* $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the set $\mathcal{A}$.[1] Actions of the sets $\mathcal{A}_1, \mathcal{A}_2, \ldots$ are performed sequentially. After performing all the actions of the set $\mathcal{A}_j$ $(j = 1, 2, \ldots)$, we perform the system test to check whether the actions contained in $\mathcal{A}_j$ have fixed the problem. The cost of $\mathcal{A}_j$ including the system test is

$$c(\mathcal{A}_j) = c_D + \sum_{A \in \mathcal{A}_j} c(A).$$

We seek an ordered partition minimimizing the ECR:

$$ECR(\mathcal{A}_1, \ldots, \mathcal{A}_k) = \sum_{i=1}^{k} c(\mathcal{A}_i) \cdot P\big(\bigwedge_{j<i, A \in \mathcal{A}_j} \{A = 0\}\big). \tag{1}$$

We shall write $P(\mathcal{A}_j = 1)$ as an abbreviation for $P\big(\bigvee_{A \in \mathcal{A}_j} \{A = 1\}\big)$. The following proposition simplifies computation of the ECR.

Proposition 2 (Ottosen and Jensen, 2010 [4]). *Under the assumptions of Proposition 1, the ECR can be computed as*

$$ECR(\mathcal{A}_1, \ldots, \mathcal{A}_k) = \sum_{i=1}^{k} c(\mathcal{A}_i) \cdot \big(1 - \sum_{j<i} P(\mathcal{A}_j = 1)\big),$$

where $P(\mathcal{A}_j = 1) = \sum_{A \in \mathcal{A}_j} P(A = 1)$.

Dynamic Programming. In [9], the authors give $\Theta(n^3)$ heuristics for *Troubleshooting with Postponed System Test* and an $\Theta(n^3 \cdot n!)$ exhaustive search algorithm. In this paragraph, we will show that by using Dynamic Programming, the time requirements of the exhaustive search can be traded for memory requirements.[2] We will use a recursive version of the definition of ECR, equivalent to the one given by Equation 1.

Definition 1 (Conditional ECR). *Let $\mathcal{A}$ be the set of available atomic actions and let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be an ordered partition of $\mathcal{A}$. For $1 \leq i \leq k$, denote*

$$\epsilon^i = \bigvee_{j \leq i, A \in \mathcal{A}_j} \{A = 0\},$$

[1] That is, $\mathcal{A} = \bigcup_{j=1}^{k} \mathcal{A}_j$, and $\forall_{i \neq j} \mathcal{A}_i \cap \mathcal{A}_i = \emptyset$.

[2] Dynamic Programming has already been used for troubleshooting in [17].

and put $\epsilon^0 = \emptyset$. *For* $1 \leq i \leq k$, *define* Conditional Expected Cost of Repair *as*

$$ECR(\mathcal{A}_i, ..., \mathcal{A}_k|\epsilon^{i-1}) = c(\mathcal{A}_i) + P(\mathcal{A}_i = 0|\epsilon^{i-1}) \cdot ECR(\mathcal{A}_{i+1}, ..., \mathcal{A}_k|\epsilon^i).$$

and put $ECR(\emptyset|\epsilon^k) = 0$.

Proposition 3 (Bellman Principle). *Ordered partition* $\mathcal{A}_1, \ldots, \mathcal{A}_k$ *is optimal if and only if each subsequence* $\mathcal{A}_i, \ldots, \mathcal{A}_k$, $1 \leq i \leq k$, *is* optimal with respect to ϵ^{i-1}, *i.e., there is no other ordered partition* $\mathbf{s}$ *defined on* $\cup_{j=i}^{k} \mathcal{A}_j$ *such that*

$$ECR(\mathbf{s}|\epsilon^{i-1}) < ECR(\mathcal{A}_i, \ldots, \mathcal{A}_k|\epsilon^{i-1}).$$

However trivial, we include proof of the "if" direction.

Proof. Assume that $\mathcal{A}_1, \ldots, \mathcal{A}_k$ has a subsequence $\mathcal{A}_i, \ldots, \mathcal{A}_k$ which is not optimal with respect to ϵ^{i-1}. In that case there exists another sequence $\mathbf{s}$, defined on $\bigcup_{j=i}^{k} A_j$, optimal with respect to ϵ^{i-1}. Concatenated sequence $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}, \mathbf{s}$ has lower ECR than the original ordered partition. □

The Dynamic Programming algorithm works in a bottom-up fashion, constructing candidate ordered partitions from the last subset. In each round, candidate subsequences that are not conditionally optimal are pruned. Correctness of the Dynamic Programming algorithm follows from Proposition 3. At the i-th round of the algorithm, 2^{n-i+1} candidates are generated, the time requirements are therefore $\Theta(2^n + 2^{n-1} + 2^{n-2} + \ldots) = \Theta(2^{n+1})$. The space requirements are dominated by the requirements of the first round, that is $\Theta(2^n)$.

2.3 Troubleshooting with Cost Clusters without Inside Information

The problem of *Troubleshooting with Postponed System Test* is a special case of *Troubleshooting with Cost Clusters without Inside Information* [7]. In the latter scenario, we assume the set $\mathcal{A}$ of atomic actions is partitioned into a family of 'cost clusters' $\{\mathcal{C}_l\}$. To access an action within the cost cluster, additional cost has to be paid for 'opening' the cluster. After opening the cluster, say $\mathcal{C}_1$, all the actions $A \in \mathcal{C}_1$ are available and actions from other clusters are not available. To access an action in a different cluster, $\mathcal{C}_1$ has to be closed and its actions are not available anymore. Furthermore, it is assumed that when any cluster is open, information about the system state is not available – the cluster has to be closed to see whether the actions taken have fixed the fault.

Solving the troubleshooting problem requires construction of an ordered partition $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of $\mathcal{A}$, where each $\mathcal{A}_j$ is a subset of some $\mathcal{C} \in \{\mathcal{C}_l\}$. For $\mathcal{A}_j \subseteq \mathcal{C}_l$, denote by $c_{\mathcal{C}(\mathcal{A}_j)}$ the cost of opening the cluster $\mathcal{C}_l$. The cost of $\mathcal{A}_j$ including the cluster cost is

$$c(\mathcal{A}_j) = c_{\mathcal{C}(\mathcal{A}_j)} + \sum_{A \in \mathcal{A}_j} c(A).$$

The ECR of troubleshooting sequence is computed as in Proposition 2.

The authors of [7] provide a heuristic algorithm for finding a suboptimal troubleshooting sequence. A related scenario, solvable in polynomial time, is studied in [8].

3 Complexity Results

We shall prove *NP*-hardness of the troubleshooting scenarios introduced in Section 2 by reducing suitable scheduling problems[3]. Moreover, we will see that the scheduling problems are equivalent to the troubleshooting scenarios in the sense that the polynomial-time reductions work both ways. Therefore, algorithms developed for the scheduling problems can be used for the troubleshooting problems without a loss of efficiency.

3.1 Reduction

We will use variants of *Single Machine Scheduling with Weighted Completion Time*. The problem is formulated as follows. There are n jobs J_i to be processed on a single machine. Each job is given with a processing time $p_i > 0$ and weight $w_i \geq 0$. We assume that processing starts at time 0 and there is no idle time between consecutive jobs. Since the processing times p_i are known and fixed, the completion time C_i of each job J_i is well determined for any given job sequence. The objective is to find a feasible sequence minimizing the *weighted completion time* $\sum_i w_i \cdot C_i$.

Single Machine Scheduling with Weighted Completion Time can easily be reduced to *Basic Troubleshooting*. Identify jobs J_i with actions A_i and put

- $p_i \longrightarrow c(A_i)$,
- $w_i / \sum_i w_i \longrightarrow P(A_i = 1)$.

The scheduling objective function can be written

$$\begin{aligned} \sum_{i=1}^{n} w_i \cdot C_i &= \sum_{i=1}^{n} w_i \cdot \sum_{j \leq i} p_i \\ &= \sum_{i=1}^{n} p_i \cdot \sum_{j \geq i} w_j. \end{aligned} \tag{2}$$

Consider the troubleshooting problem. Assuming $P\big(\vee_{A \in \mathcal{A}} \{A = 1\}\big) = 1$ under the conditions of Proposition 1, we use Proposition 1 and rewrite the definition of ECR:

$$\begin{aligned} ECR(A_1, \ldots, A_n) &= \sum_{i=1}^{n} c(A_i) \cdot \big(1 - \sum_{j<i} P(A_j = 1)\big) \\ &= \sum_{i=1}^{n} c(A_i) \cdot \sum_{j \geq i} P(A_j = 1). \end{aligned} \tag{3}$$

It is clear that Equation 3 is minimized whenever Equation 2 is minimized. □

In analogy to Proposition 1, the *Single Machine Scheduling with Weighted Completion Time* problem can be solved by sequencing the jobs in non-increasing order of the ratios w_i/p_i. This result can be traced back to a 1950's paper by Smith [12].

[3] See [3] for an overview of the field.

3.2 Troubleshooting with Cost Clusters without Inside Information

We reduce *Single machine s-batching with weighted completion time* [1,3]. As above, there are n jobs given with processing time p_i and weight w_i. The jobs are scheduled in *batches* on a single machine. A batch is a set of jobs which are processed jointly. Processing time of a batch equals the sum of processing times of its jobs plus a *batch setup time* s. *Completion time* C_i of a job coincides with the completion time of the last scheduled job in its batch (completion times of all jobs in a batch are therefore equal). The task is to find a sequence of jobs and partition it into batches such that we minimize $\sum_{i=1}^{n} w_i \cdot C_i$. We sum up properties of the batching problem in a proposition.

Proposition 4 (Albers and Brucker, 1993 [1]). *Single machine s-batching with weighted completion time is* NP*-hard. Given a fixed sequence of jobs, the split into batches can be done in* $O(n)$ *time. When* $w_i = 1$ *or* $p_i = p$ *for all* i*, the problem becomes solvable in* $O(n \log n)$ *time.*

Proposition 5. *Troubleshooting with Postponed System Test is* NP*-hard, even under the assumptions of Proposition 1.*

Proof. We will use a description of solutions of the batching problem taken from [3]. Consider a fixed but arbitrary job sequence $J_1, \ldots, J_n$. Denote the batch setups by S. Then the solution takes on the form

$$S, J_{\lambda(1)}, \ldots J_{\lambda(2)-1}, S, J_{\lambda(2)}, \ldots, J_{\lambda(k)-1}, S, J_{\lambda(k)}, \ldots, J_n$$

where k is the number of batches, $\lambda(j)$ is the starting index of the j-th batch, and

$$1 = \lambda(1) < \lambda(2) < \ldots < \lambda(k) \leq n$$

The processing time of the j-th batch is

$$P_j = s + \sum_{\ell=\lambda(j)}^{\lambda(j+1)-1} p_\ell.$$

The objective function can now be written as

$$\sum_{i=1}^{n} w_i \cdot C_i = \sum_{j=1}^{k} P_j \cdot \sum_{\ell=\lambda(j)}^{n} w_\ell \tag{4}$$

We proceed with the reduction as in the beginning of Section 3.1. Using Proposition 2 and assuming $P\big(\bigvee_{A \in \mathcal{A}} \{A = 1\}\big) = 1$, we rewrite

$$ECR(\mathcal{A}_1, \ldots, \mathcal{A}_k) = \sum_{j=1}^{k} c(\mathcal{A}_i) \cdot \sum_{\ell \geq i} P(\mathcal{A}_\ell = 1). \tag{5}$$

The correspondence of Equations 5 and 4 is obvious. □

We can also easily perform the reduction in the opposite direction. Therefore, Proposition 4 applies to *Troubleshooting with Postponed System Test* (under the assumptions of single fault, and actions conditionally independent given faults). In particular, the $O(n)$ bound on partitioning a fixed sequence of actions is an improvement over $\Theta(n^3)$ given in [9].

Corollary 1 (of Proposition 5). *Troubleshooting with Cost Clusters without Inside Information is* NP*-hard, even under the assumptions of Proposition 1.*

Proof. Consider a troubleshooting problem where all the actions belong to a single cost cluster $\mathcal{C}_1$ with the cost of opening $\mathcal{C}_1$ being c_D. This problem is equivalent to *Troubleshooting with Postponed System Test.* □

Remark 1. The *decision variants* of all the troubleshooting scenarios studied in this paper clearly belong to *NP* – a nondeterministic procedure can guess a troubleshooting sequence and then check in polynomial time whether the *ECR* is lower than a predefined constant. Therefore, by Proposition 5 and Corollaries 1 and 2, the decision variants of the respective troubleshooting scenarios are *NP*-complete.

3.3 Troubleshooting with Precedence Constraints

We now introduce *Single Machine Scheduling with Weighted Completion Time and Precedence Constraints* [6]. The problem is the same as *Single Machine Scheduling with Weighted Completion Time*, with an additional requirement that the sequencing of the jobs has to be consistent with precedence constraints imposed by a given acyclic directed graph $G = (V, E)$. Each vertex $i \in V$ is identified with a job. Job J_i is to precede job J_j if there is a directed path from i to j in G.

Proposition 6 (Lawler, 1978 [6]). *Single Machine Scheduling with Weighted Completion Time and Precedence Constraints is* NP*-complete, even if all* $w_i = 1$ *or all* $p_i = 1$.

Corollary 2. *Basic Troubleshooting with Precedence Constraints is* NP*-hard, even under the assumptions of Proposition 1.*

Proof. Use the reduction from Section 3.1. □

Next, we define a class of series parallel directed graphs for which *Single Machine Scheduling with Weighted Completion Time and Precedence Constraints* is known to be polynomial. This class subsumes chains and rooted trees. As such, it is quite useful for applications.

Definition 2 (*MSP – Minimal Series Parallel Graph*, [15]). *The graph consisting of a single vertex and no edges is MSP.*

If directed graphs $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *are MSPs, so is either of the directed graphs constructed by the following operations:*

- Parallel composition*:* $G = (V_1 \cup V_2, E_1 \cup E_2)$.
- Series composition*:* $G = (V_1 \cup V_2, E_1 \cup E_2 \cup N_1 \times R_2)$. *Here* N_1 *is the set of* sinks *of* G_1 *(i.e., vertices without successors), and* R_2 *is the set of* sources *of* G_2 *(i.e., vertices without predecessors).*

Recall the concept of *transitive closure* – given a directed graph $G = (V, E)$, its transitive closure $G_T = (V, E_T)$ is obtained by adding a directed edge (u, v) for all u and v such that there is a path from u to v in G and $(u, v) \notin E$. A *transitive reduction* of $G = (V, E)$ is a minimal graph G_R defined on V such that the transitive closures of G and G_R are the same.

Definition 3 (*GSP – General Series Parallel Graph*, [15]). *A directed graph is GSP if its transitive reduction is an MSP.*

Proposition 7 (Lawler, 1978 [6]). *Single Machine Scheduling with Weighted Completion Time and Precedence Constraints is solvable in* $O(n \cdot \log n)$ *time when the precedence graph* G *is a GSP.*

Reversing the reduction, we get the following easy consequence.

Corollary 3. *Basic Troubleshooting with Precedence Constraints is solvable in* $O(n \cdot \log n)$ *time under the following conditions:*

- *all the assumptions of Proposition 1 are satisfied,*
- $P\big(\vee_{A \in \mathcal{A}} \{A = 1\}\big) = 1$,
- *the precedence graph is a GSP.*

4 Conclusions and Future Research

We have established a link to the well developed field of Scheduling, opening the possibility of applying results of decades of research in Scheduling to Troubleshooting. We have reduced scheduling problems to derive proofs of *NP*-hardness for three troubleshooting scenarios. The scenario of *Basic Troubleshooting with Precedence Constraints* is novel. We believe this scenario is useful in practice. Moreover, it is polynomial for a wide class of graphs encoding the precedence relation.

An interesting problem for future research is that of *approximability* [10] of troubleshooting problems: for some special cases there might exist approximation algorithms with performance guarantees, whereas for others, finding such an approximation would amount to proving $P = NP$.

Since most realistic troubleshooting scenarios are *NP*-hard, it is worthwhile to study heuristic solution algorithms [17,7,9] and identify worst-case conditions, under which they perform badly. To benchmark the heuristic algorithms, we can use Dynamic Programming introduced in Section 2.2.

Acknowledgments. I would like to thank Thorsten J. Ottosen and Jiří Vomlel for valuable discussions over the subject matter of the paper. I also thank the anonymous reviewers for their comments.

References

1. Albers, S., Brucker, P.: The Complexity of One-Machine Batching Problems. Discrete Applied Mathematics 47, 87–107 (1993)
2. Breese, J.S., Heckerman, D.: Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment. In: Proceedings of Twelfth Conference on Uncertainty in Artificial Intelligence, pp. 124–132. Morgan Kaufmann, San Francisco (1996)
3. Brucker, P.: Scheduling Algorithms, 3rd edn. Springer, Heidelberg (2001)
4. Jensen, F.V., Kjærulff, U., Kristiansen, B., Langseth, H., Skaanning, C., Vomlel, J., Vomlelová, M.: The SACSO Methodology for Troubleshooting Complex Systems. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 15, 321–333 (2001)
5. Kalagnanam, J., Henrion, M.: A Comparison of Decision Analysis and Expert Rules for Sequential Diagnosis. In: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence (UAI 1988), pp. 271–282. North-Holland, Amsterdam (1990)
6. Lawler, E.L.: Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints. Annals of Discrete Mathematics 2, 75–90 (1978)
7. Langseth, H., Jensen, F.V.: Heuristics for Two Extensions of Basic Troubleshooting. In: Proceedings of Seventh Scandinavian Conference on Artificial Intelligence, SCAI 2001, pp. 80–89. IOS Press, Amsterdam (2001)
8. Ottosen, T.J., Jensen, F.V.: The Cost of Troubleshooting Cost Clusters with Inside Information. In: Proceedings of 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010), pp. 409–416. AUAI Press (2010)
9. Ottosen, T.J., Jensen, F.V.: When to Test? Troubleshooting with Postponed System Test. Technical Report 10-001, Department of Computer Science, Aalborg University, Denmark (2010)
10. Papadimitriou, C.H.: Computational complexity. Addison-Wesley Publishing Company, Reading (1994)
11. Sedgewick, R.: Algorithms in C. Addison-Wesley Publishing Company, Reading (1998)
12. Smith, W.E.: Various Optimizers for Single-Stage Production. Naval Research Logistics Quarterly 3, 59–66 (1956)
13. Skaanning, C., Jensen, F.W., Kjærulff, U.: Printer Troubleshooting Using Bayesian Networks. In: IEA/AIE 2000 Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 367–379. Springer, Heidelberg (2000)
14. Srinivas, S.: A Polynomial Algorithm for Computing the Optimal Repair Strategy in a System with Independent Component failures. In: Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence, pp. 515–552. Morgan Kaufmann, San Francisco (1995)
15. Valdes, J., Tarjan, R.E., Lawler, E.L.: The Recognition of Series Parallel Digraphs. In: STOC 1979 Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, pp. 1–12. ACM, New York (1979)
16. Vomlelová, M.: Complexity of Decision-Theoretic Troubleshooting. International Journal of Intelligent Systems 18, 267–277 (2003)
17. Vomlelová, M., Vomlel, J.: Troubleshooting: NP-hardness and Solution Methods. Soft Computing Journal 7(5), 357–368 (2003)