FlowOpt: Bridging the Gap Between Optimization Technology and Manufacturing Planners

Roman Barták[1*], Milan Jaška[1], Ladislav Novák[1], Vladimír Rovenský[1], Tomáš Skalický[1], Martin Cully[2], Con Sheahan[2], Dang Thanh-Tung[2]

[1] Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, Praha, Czech Republic

[2] Entellexi Ltd., National Technology Park, Limerick, Ireland

[*] bartak@ktiml.mff.cuni.cz (contact e-mail)

Abstract

FlowOpt is demonstration software showing how advanced optimization technology can be put at the fingertips of practitioners, who are not experienced in optimization. FlowOpt is a unique combination of optimization technology based on constraint satisfaction techniques with knowledge engineering aspects of extracting formal problem models and with interactive visualization that gives users full control over the produced schedules. The system supports visual design of production workflows with alternative processes, it generates feasible schedules based on the workflows and specified resources, it shows the schedules in the form of a Gantt view with the possibility to manually modify the schedules, and finally, it supports analysis of schedules to find new ways to improve production.

Introduction

One of the biggest problems of today's advanced technology is its limited accessibility to users working in a given domain, but not necessarily being experts in the used technology. Apple's iPhone is a great example how the advanced technology can be made accessible to regular users. With the tradeoff of slightly limited functionality, it provides user interface to very advanced techniques such as Q&A (question and answering) that anyone can immediately use without the hassle of long training.

Though there exists a vast amount of research in the area of scheduling, there is still a large gap between practical problems and research results especially in the area of production optimization for small and medium enterprises (SMEs). This gap is partly due to missing modeling and visualization tools that would allow easy transformation of real-life problems to optimization models and the results back to customers (Barták et al. 2010) and partly due to large distance of academic optimization algorithms from the existing problems.

Competition in the area of production is increasing due to globalization of the production market and entrance of low-wage economies to this area. Many large companies are outsourcing or moving their production facilities to low-wage countries as an easy way to decrease the production costs. This approach is not applicable to small and medium enterprises (SMEs) that are typically family-based and connected to their area of origin. These companies can survive in the market by providing high quality products that are built-to-order or even engineered-to-order to fit exactly the specific needs of the customers. Production of a large variety of items in small quantities is very demanding on work organization especially when deadlines are tight. This is becoming a serious problem for SMEs where production planning is frequently done manually using Excel sheets. Existing production optimization tools are built mainly for larger companies and the tools require serious tuning to a particular production facility. Moreover, such tools and related services are too expensive for SMEs. Hence there is a need for an easy-to-use optimization tool that bridges the gap between the real-world problems and advanced optimization technology. The tool must be easily customized to most production facilities and it must be easy-to-use by people that are not experts in optimization.

FlowOpt is a system that attempts to address the above problem and bridges the gap between advanced optimization technology developed at universities and practitioners from production planning. FlowOpt is a software tool targeted to discrete manufacturing SMEs. It is intended as a technology demonstration showing how current optimization techniques can be used in practice. The system is customizable to a particular production facility. The users first need to describe the production workflows for their products. Then they tell the system what products should be manufactured and when they should be finished, and the system generates

a feasible production plan/schedule. The plan is visualized to the user, who can do any modification of the plan. This is a critical feature, as the users need to have a full control over the production plans. Based on the generated schedule, the system can also analyze the company and recommend how to change it to increase efficiency (for example by buying a new machine). The human computer interaction is done via intuitive graphical interface hiding the complexity of the optimization process. FlowOpt is a student software development project at Charles University in Prague. The software itself is a collection of closely interconnected plug-in modules to the enterprise performance optimization system MAK€ from Entellexi Ltd. (Barták et al. 2012).

This paper describes the capabilities and technology behind the FlowOpt system. We will first introduce the core modeling concepts, namely the nested workflows extended by additional constraints. Then we will go through the production planning process as it is realized in the FlowOpt system. Finally, we will give more details about the AI technology used inside the system. The paper is concluded by the discussion of possible future extensions both from the application and research perspectives.

Formal Background

FlowOpt is a production planning/scheduling system for SMEs. Unlike many similar systems that were designed to optimize production in particular factories, the initial design concept behind the FlowOpt system was to support various production facilities. Hence the system must allow the users to describe the production environment. The FlowOpt system is built around the concept of workflow that is an organized collection of activities to achieve some goal. In our case the goal is the production of certain item and there are precedence and causal relations between the activities. Using the idea of workflow gives us the flexibility to describe virtually any production process and hence the system can optimize any production facility that can be modeled there. Note that though there exist other attempts to generally describe production processes for example using the traditional planning view via preconditions and effects (Do et al. 2011). The structure of workflows has the advantage of better organization of production activities that is closer to the production view. The additional structural information within the workflow can also be exploited in the optimization procedure similarly to exploiting the structure in hierarchical task networks (HTN). The idea of exploiting the hierarchical structure of workflows was independently explored in the JABBAH system where the Business Process Models are directly translated to HTN (González-Ferrer et al. 2009).

Nested Workflows

There exist many formal models of workflows such as BPMN (Business Process Modeling Notation) or YAWL (Yet Another Workflow Language)[1]. These are very generic models that support repetition of parts of the workflow and many specific constraints between the activities. In the FlowOpt system we adapted the idea of Nested Temporal Network with Alternatives (Barták and Čepek 2008). Temporal Network with Alternatives (TNA) is a directed acyclic graph with parallel and alternative splitting and joining of processes that is also known as AND-split and OR-split (and AND-join, OR-join) in traditional workflow management systems (Bae et al. 2004). Hence TNA can describe alternative processes in the style similar to scheduling workflows with optional activities introduced in (Beck and Fox 2000) and used also in Extended Resource Constrained Project Scheduling Problems (Kuster et al. 2007). Nested TNA restricts the structure of TNA in such a way that each split operation has a corresponding join operation. This idea was informally described in (Beck and Fox 2000) and formalized in (Barták and Čepek 2008) and it seems that it is quite common in real-life workflows (Bae et al. 2004). This restriction brings a hierarchical structure to workflows very similar to hierarchical task networks leading

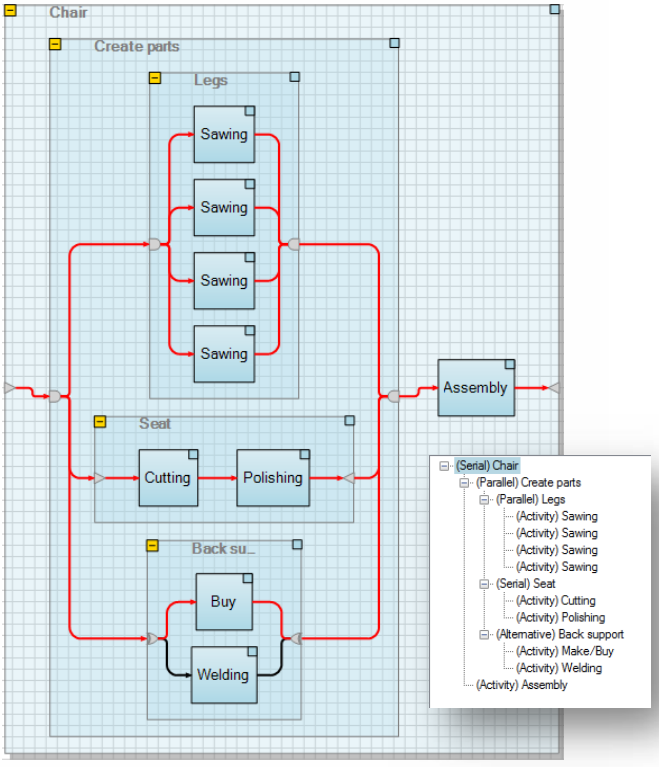

Figure 1. Visualization of a nested workflow in the FlowOpt Workflow Editor (from top to bottom there are parallel, serial, and alternative decompositions)

[1] http://www.bpmn.org/, http://www.yawlfoundation.org/

to Temporal Planning Networks (TPN) (Kim et al. 2001). From the manufacturing perspective, the hierarchical structure resembles the concept of *bill of materials*.

In the FlowOpt system we use a slightly modified view of Nested TNA called a *nested workflow*. A nested workflow is obtained from a root task by applying decomposition operations that split the task into subtasks until primitive tasks, corresponding to activities, are obtained. Three decomposition operations are supported, namely parallel, serial, and alternative decomposition. Figure 1 gives an example of a nested workflow that shows how the tasks are decomposed. The root task *Chair* is decomposed serially into two tasks, where the second task is a primate task filled by activity *Assembly*. The first task *Create Parts* decomposes further to three parallel tasks *Legs*, *Seat*, and *Back Support*. *Back Support* is the only example here of alternative decomposition into two primitive tasks with *Buy* and *Welding* activities (*Welding* is treated as an alternative to *Buy*). Naturally, the nested workflow can be described as a tree of tasks. This is similar to hierarchical task networks though the nested workflows do not support repetition of tasks and the number of tasks is fixed there.

Formally, the nested workflow is a set *Tasks* of tasks, that is a union of four disjoint sets: *Parallel*, *Alternative*, *Serial*, and *Primitive*. For each task T (with the exception of the root task), function *parent*(T) denotes the parent task in the hierarchical structure. Similarly for each task T, we can define the set *subtasks*(T) of its child nodes (*subtasks*(T) = {C ∈ *Tasks* | *parent*(C) = T}). The tasks from sets *Parallel*, *Alternative*, and *Serial* are called *compound tasks* and they must decompose to some subtasks:

$$T \in (Parallel \cup Alternative \cup Serial) \Rightarrow subtasks(T) \neq \emptyset,$$

while the primitive tasks do not decompose:

$$T \in Primitive \Rightarrow subtasks(T) = \emptyset.$$

The workflow defines one or more processes in the following way. *Process* selected from the workflow is defined as a subset $P \subseteq Tasks$ in the workflow satisfying the following properties:

- for each task T in the process, that is not a root task, its parent task is also in the process,
  $T \in P \wedge T \neq root \Rightarrow parent(T) \in P$,
- for each compound task T in the process with a serial or parallel decomposition, all its subtasks are also in the process,
  $T \in P \cap (Serial \cup Parallel) \Rightarrow subtasks(T) \subseteq P$,
- for each compound task T in the process with the alternative decomposition, exactly one of its subtasks is in the process,
  $T \in P \cap Alternative \Rightarrow |\, subtasks(T) \cap P \,| = 1$.

So far we defined only the hierarchical structure of the nested workflow, but as Figure 1 shows the nested structure also defines several temporal constraints. These temporal relations must hold for all tasks in a single process. Assume that S_T is the start time and E_T is the completion time of task T. The primitive tasks T are filled by activities and each activity has some duration D_T. Then for tasks in certain process P the following relations hold:

$$T \in P \cap Primitive \Rightarrow S_T + D_T = E_T$$

$$T \in P \cap (Parallel \cup Alternative \cup Serial) \Rightarrow$$
$$S_T = \min\{ S_C \mid C \in P \cap subtasks(T) \}$$
$$E_T = \max\{ E_C \mid C \in P \cap subtasks(T) \}.$$

Notice that the duration of compound task is defined by the time allocation of its subtasks while the durations of primitive tasks are defined by the activities. Moreover, for the serial decomposition we expect the subtasks to be ordered, say $T_1, \ldots, T_n$, where n is the number of subtasks of a given task. The following constraints must hold, if the serial task is included in the process:

$$\forall i = 1,\ldots, n\text{-}1: E_i \leq S_{i+1}$$

A *feasible process* is a process where the time variables S_T and E_T can be instantiated in such a way that they satisfy the above constraints. It is easy to realize that if there are no additional constraints then any process is feasible. The process defines a partial order of tasks so their start and end times can be easily set in the left-to-right order while satisfying all the above constraints.

Extra Constraints

The core nested structure has the nice property that for each task there exists a feasible process containing that task (Barták and Čepek 2008). However, the nested structure may not be flexible enough to describe naturally some additional relations in real-life processes, for example when the selected alternative for one task influences the selection of alternatives in other tasks. Therefore in the FlowOpt we support specification of extra binary constraints between the tasks:

- *precedence constraints* between any pair of tasks mean that if both tasks are selected in the process then the specified temporal ordering must hold,
- *temporal synchronization constraints* describe that two tasks start or end at the same time, or that one task must start exactly when another task finishes,
- *logical constraints* describe the causal relations between the tasks beyond the nested structure; mutual exclusion requires that both tasks cannot appear at a single process, equivalence constraint requires that either both tasks appear in the same process or none of them, and implication constraint A ⇒ B requires that if task A appears in the process then also task B must appear in the same process.

These constraints must also be satisfied in the feasible process. Let us now describe them formally for any two tasks i and j:

- precedence constraint: $i, j \in P \Rightarrow E_i \leq S_j$
- start-start synchronization: $i, j \in P \Rightarrow S_i = S_j$
- start-end synchronization: $i, j \in P \Rightarrow S_i = E_j$
- end-start synchronization: $i, j \in P \Rightarrow E_i = S_j$
- end-end synchronization: $i, j \in P \Rightarrow E_i = E_j$
- mutex constraint: $i \notin P \vee j \notin P$
- equivalence constraint: $i \in P \Leftrightarrow j \in P$
- implication constraint: $i \in P \Rightarrow j \in P$

It is interesting that if these extra constraints are used then the existence of a feasible process is no more obvious. In fact, the problem of deciding whether a feasible process exists for a nested workflow with extra constraints is NP complete (Barták 2012). The consequence is that when extra constraints are used then there is no guarantee that there is a feasible process for each task in the workflow. There might be such tasks than cannot be used in any feasible process; for example, the extra precedence constraints may add a cycle. Therefore, it is useful to verify the workflows and report such cases to the user (Giro 2007), because the task that cannot be used in any feasible process indicates a serious design flaw.

Resources

In addition to duration, each activity is also accompanied by a set of resources that it requires for processing. The resource can be a machine, a tool, or a worker; we do not use any particular restriction on the type of resource. The activity must be allocated to all resources that it requires. The FlowOpt system currently supports only so called *unary* (*disjunctive*) *resources*. Unary resource is a resource that can process at most one activity at any time. We assume the activities to be *non-interruptible*, that is, when we start processing the activity, we must continue until its finished. Let *res*(T) be the set of resources required by a primitive task T (activity inside it) and S be a set of tasks to be scheduled. Then the limited capacity of the resource can be described formally using the disjunctive constraint:

$$i, j \in S \cap Primitive \wedge res(i) \cap res(j) \neq \emptyset \Rightarrow E_i \leq S_j \vee E_j \leq S_i.$$

This constraint basically says that if two primitive tasks share a resource then these tasks must be ordered somehow, either *i* before *j* or vice versa. If these constraints are satisfied by tasks from S then we say that S is *resource feasible*. We use the set S instead of P because we now assume a set of feasible processes selected from the workflows to be allocated to shared resources. This is the *scheduling problem* that the FlowOpt system solves: we have a set of workflows at the input and the task is to select a feasible process P_i for each workflow *i* such that the set $S = \cup_i P_i$ is resource feasible. This problem is similar to Resource Constrained Project Scheduling Problem (RCPSP) with the extension that we need to select the tasks to be scheduled from the workflows.

Objectives

Users are usually looking for good schedules not only for feasible schedules. There exists many criteria to evaluate quality of schedules; makespan is probably the most famous one though not the most practically relevant. In the current version of FlowOpt we use a criterion called *on-time-in-full*. The user selects the set of workflows to be scheduled, the start time, and the due date for the schedule. It is also possible to specify whether the workflows can finish earlier or later with respect to the due date and in such a case what penalty cost is paid (per time unit). The task is to find a resource feasible schedule with the minimal penalty.

FlowOpt Functionality

One of the primal tasks of the FlowOpt system is to simplify problem modeling for the end users by hiding the complexity of mathematical formalism while guiding the users to design correct and complete models. The FlowOpt system is a collection of five modules that are responsible for individual steps in the process of production optimization. The general purpose of the FlowOpt system is to provide a streamlined feature-rich environment where the user can do the following activities in a simple, efficient, and user-friendly way:

1. specify how a particular product is manufactured, i.e., define a workflow describing the manufacturing of a single product,
2. describe the work order from the customers – the work order is specified by the products, their quantities, the start time of production, and the desired due date,
3. generate a schedule for the order – the schedule is a complete description of which tasks should be performed, when they should be performed, and what resources they should use; executing such a schedule results in efficient fulfilling of the work order,
4. display the generated schedule in the form of a Gantt chart and allow the user to modify it interactively,
5. analyze the generated schedule and suggest opportunities for improvements of the factory.

FlowOpt is not a standalone product. It is built on top of the commercial system MAK€ that is a performance prediction and optimisation tool for SMEs (Barták et al., 2012). MAK€ already provides tools such as Workflow Editor, Optimizer, and Gantt Viewer, and the goal of FlowOpt is to demonstrate a new functionality for these modules. By building the FlowOpt system on top of MAK€, we can exploit some existing functionality such as the database for storing workflows and schedules and the resource and activity manager. Hence we assume that we already have a set of primitive activities that the factory can process and a set of available resources (people, machines, tools, etc.). In

this section we will describe the features of individual FlowOpt modules from the user perspective.

Workflow Editor

Describing the workflow structure is the first step in specifying how the factory works (after describing the primitive production activities and resources that are taken from the MAK€ system). Visual editing of workflows is not a new idea, YAWL (van der Aalst and Hofstede 2005) and MAK€ (Barták et al. 2012) already provide graphical editors for describing workflows. These editors look like drawing programs, where the user puts activities on a virtual board and connects them via constraints that describe the flow of material (precedence constraints) and other relations between the activities. This is a very flexible process and virtually any workflow can be drawn this way. However, the tradeoff is that the user is also responsible for everything starting from the arrangement of items on the board to make the workflow visually acceptable and concluding with ensuring that the internal structure of the workflow is flawless. Though the verification can sometimes be done automatically after the workflow is drawn, we believe that a much better approach is guiding the user to draw as flawless workflows as possible.

As we mentioned at the beginning we decided to go in the "iPhone" way with a slightly restricted flexibility but with much higher user comfort in modeling the workflows. First, rather that drawing any graph to describe the workflow and then checking whether the graph corresponds to some specific structure (such as the nested structure) we guide the user to design the workflows only with the nested structure. This can be done be constructing the workflow using the decomposition operations described in the previous section. Basically, when designing a new workflow, we start with a root task and the user can apply any of the three decomposition operations to this task and to the obtained subtasks. It is also possible to drag an action from the list of actions to the non-decomposed task to include the action in the workflow. We expect that the list of actions with their durations and required resources is given. Actually, we use the set of actions from the underlying MAK€ system.

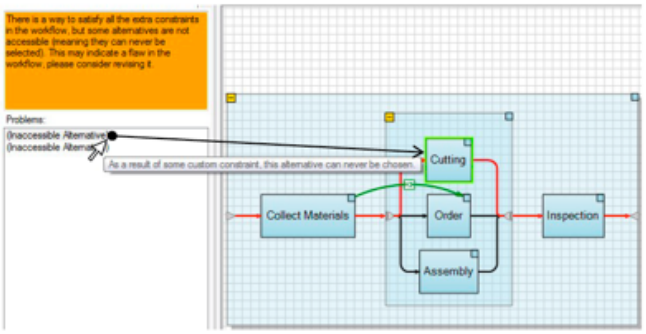

Figure 2. Highlighting the found flaws after workflow verification

The *top-down approach* is useful, if the users have already quite an organized view of what they are doing. This is not always the case in SMEs where they need first to organize their processes into workflows. Hence, the FlowOpt Workflow editor also supports a *bottom-up approach* to building workflows that is closer to the traditional workflow editors. The users put actions on the board but rather than connecting them with lines, the users are grouping them. Grouping is a reverse operation to decomposing and the actions can be grouped to form a sequence, to form a parallel task, or to form a set of alternatives. This approach seems more appropriate when describing existing processes, and it is a form of knowledge extraction. This way the user is guided to describe some structure of the workflow. The top-down and bottom-up approaches can be used together; the workflow designed by the bottom-up approach can be, for example, placed to an existing task in the hierarchical structure by the drag-and-drop operation. The only restriction is that at the end we should obtain a single workflow that is inside the root task. Whether the workflow is designed by the decomposition operations or composed from the activities is up to the user. The editor also takes care of the alignment of sub-tasks in the tasks and of their spatial organization on the board. The user only focuses on the structural organization of the operations into tasks. Figure 1 shows how the workflow looks like in the editor if the workflow is organized left-to-right (top-down organization is also possible).

All above decomposition and composition operations can be done using intuitive drag-and-drop operations. The extra binary constraints, described in the previous section, can also be added using drag-and-drop operations. The user just draws a line connecting one task with another task and selects a particular type of the constraint. Adding these extra constraints could introduce flaws into the workflow. The editor can detect some of these flaws immediately and forbid adding the constraint to the workflow. However, as the problem of workflow verification is NP-complete (Barták 2012) it is not possible to detect all flaws during the workflow design. Therefore the editor includes a function that can verify the workflow on demand. Though the process of workflow verification is quite complex (Rovenský 2011), we succeeded to hide this complexity from the user, and the verification is as easy as pushing a button in GUI (graphical user interface). If any flaws are found, they are reported to the user and also highlighted in the visual model of the workflow together with the textual description of the flaw (Figure 2).

As the FlowOpt Workflow Editor is build on top of the MAK€ system, which has its own workflow editor, it is possible to import existing workflows from MAK€ to the FlowOpt system. This process requires the original unstructured workflows to be converted to the nested structure (Barták and Čepek 2008), which is not always

possible, so only nested workflows can be imported. This way we allow the customers to easily switch to the FlowOpt system. JABBAH (González-Ferrer et al. 2009) provided similar functionality with the more general BPM workflows converted to the HTN formalism.

Order Manager and Optimizer

The workflows describe what products can be made in the factory and how they are produced. In the AI planning terminology, they describe the planning domain. To obtain an actual production plan the user should select the items to be produced and the time horizon when the production should happen. Recall that each item corresponds to one workflow, so the user basically selects the workflows and specifies how many times each workflow appears in the schedule (how many items will be produced). In the AI planning terminology it corresponds to selecting the tasks to be achieved. Additionally, the user describes the start time, that is, the earliest time when any operation can start, and specifies the due date when the production should be finished. The due date can be set as a hard deadline that cannot be exceeded or as a soft deadline where exceeding it is penalized by some fix cost per time unit. In the second case we are solving an optimization problem. These data are then passed to the optimizer that produces a production plan, where operations are selected from the workflows and allocated to particular times and resources. This is a scheduling problem with some planning component of selecting alternatives from the workflows. The Optimizer is basically trying to find some production plan, and then to improve its quality based on the earliness and lateness cost. The status of the planner is reported to the user using a familiar progress bar indicating whether the Optimizer has already found a solution and is improving it, or is still looking for the first feasible production plan. The user can interrupt the Optimizer at any time and obtain the best production plan found so far (if any). This is an important feature as the users can trade-off the plan quality for the runtime.

Gantt Viewer

The generated schedule (production plan) can be visualized in the Gantt Viewer. This module provides both traditional views of the schedule, namely the task-oriented and resource-oriented views. Gantt charts are a classical way how to visualize schedules so let us focus on the unique features of the FlowOpt Gantt Viewer. First, the Gantt Viewer has full access to the workflow specification, not only to the production plan outputted from the Optimizer. Having this complete information, the Gantt Viewer can visualize the alternatives that were not selected by the Optimizer. It also allows users to modify any aspect of the production plan using drag-and-drop techniques. The user can move activities to different times and resources and change their duration. It is even possible to select another alternative than the one suggested by the Optimizer. Because the Gantt Viewer is aware of all the constraints originating from the workflow specification, it can also highlight violation of any of these constraints. Obviously, the production plan generated by the Optimizer is flaw-less and no constraint is violated, but the user modifications may introduce flaws such as breaking a precedence constraint or exceeding the capacity of some resource. The Gantt Viewer highlights these flaws so it is easy to spot them (Figure 3).

Another unique feature of the Gantt Viewer is the capability to repair the schedule. Currently the system repairs the schedule by shifting activities in time while trying to keep the production plan as close as possible to the original plan (Barták and Skalický, 2009). Similar to verification, this complex process of re-scheduling is hidden from the user and it is initiated by pressing a single button in GUI.

Analyzer

The final module is Analyzer that is responsible for suggesting improvements of the production process. The Analyzer takes an existing schedule and finds bottlenecks in the schedule such as an activity that delays the delivery of the final item. The bottlenecks are presented to the user who may select some of them for further processing or even add additional bottlenecks that were not identified automatically. For each selected bottleneck, the analyzer suggests how to resolve it – this could be for example by buying a new resource or by decreasing the duration of certain activities (for example by staff training). This is called an *improvement project* and again the user can add them manually. In the third step, the improvement projects are evaluated using the Optimizer. The goal of evaluation is finding the real value of the improvement project, that is, how much it can improve the current schedule, and also finding possible relations between the improvement projects. For example, the system can detect that if two improvement projects are applied together then the overall improvement is much better than the sum of improvements of the individual projects. After manual adding of cost to each improvement project the system selects a subset of improvement projects such that their combination brings the best overall improvement of the production process under the given constraints such as a limited budget to realize the improvement projects.

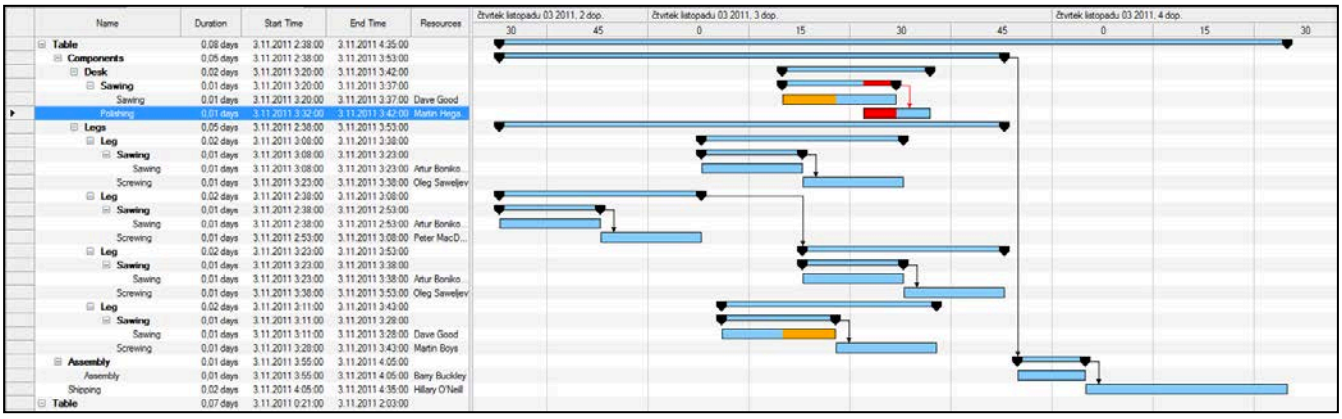

Figure 3. The task view of the FlowOpt Gantt Viewer with two highlighted constraints that are violated (exceeded capacity of resource Dave Good and broken precedence constraint).

The FlowOpt Analyzer is a form of mixed-initiative decision support system where the user can ask the system to provide solutions as well as to evaluate proposals suggested by the user. In each step of the improvement process, the system generates some suggestions, such as the improvement projects, the user can modify them or even manually add new improvement projects, and ask the system to evaluate only the selected projects and then build a portfolio from them. It should be said that the Analyzer is not improving the schedule based on the fixed problem specification; this is the task of the Optimizer. The Analyzer modifies the problem specification to obtain better schedules, which is a unique feature not present in other systems.

FlowOpt Technology at Glance

FlowOpt is a complex system that combines advanced user interfaces with sophisticated optimization technology. Surprisingly, the most complex algorithms are the least visible components in the user interface. In some sense this was the intention of the FlowOpt system that is supposed to bring the advanced techniques to regular users.

In this section we will focus on the technology that is behind the FlowOpt system. First, it is important to say that the FlowOpt system is a general production planning system that is supposed to solve any particular problem formulated in the provided formalism. Hence, it is not possible to use ad-hoc scheduling algorithms and we need a more general solving technology. We decided to use *constraint satisfaction* as it can be used for any combinatorial optimization problem while it can be customized for particular problems by choosing the right constraint model. In fact, the constraint satisfaction techniques are used in every module of the FlowOpt system.

Workflow Verification

One of the tasks of the FlowOpt Workflow Editor is verification of workflows, that is, checking that every task can be used in some process selected from the workflow. This is guaranteed for the nested workflows, but when the extra constraints are added, the verification problem becomes NP-complete (Barták, 2012). Hence there is a little hope for the polynomial algorithm so we developed a verification technique similar to one from (Bi and Zhao 2004) but using constraint satisfaction. Briefly speaking, we model the problem of selecting a process from the workflow as a constraint satisfaction problem. Each task is modeled using a Boolean validity variable indicating whether the task is selected to the process or not. There are also temporal variables describing the start and end times of tasks (resource constraints are not assumed during verification). The constraints are derived directly from the description of the workflow (see the formal model). Now the problem is to find out if for each validity variable, there exists a solution to the above-sketched CSP, where the validity variable is set to value true. This corresponds to finding a valid process for a given task. As we are using some specific temporal consistency techniques, namely Incremental Full Path Checking (Planken, 2008), and also a specific search strategy that remembers, if any validity variable was set to true, so there is no need to check it again, we decided to implement the verification algorithm from scratch rather than using an existing constraint solver (Rovenský 2011).

Scheduling

The Optimizer solves a *scheduling problem* with optional activities, so it was again natural to use constraint satisfaction. For the Optimizer we decided to use an existing constraint satisfaction package, namely ILOG CP Optimizer, because there is a strong support for scheduling constraints (Laborie, 2009). Moreover, the CP Optimizer already supports so-called optional activities and the

hierarchical structure of tasks (Laborie and Rogerie, 2008). Hence we can fully exploit the existing constraints in the CP Optimizer and its built-in search strategy; no special solving algorithm was used. The result of scheduling is identification of selected operations that are allocated to precise time and required resources. Solving the optimization problem is realized via the branch-and-bound procedure where the initial feasible solution is generated first and better solutions are generated next using the same mechanism. The innovative part here is fully automated generation of the constraint model from the description of workflows, activities, and resources using the techniques from (Barták et al. 2010).

Re-Scheduling

The Gantt Viewer also contains some form of scheduling when we need to repair the violated constraints after manual modification of the schedule. We extended the re-scheduling algorithm proposed in (Barták and Skalický 2009) to cover all the constraints from the current problem. Again, the technology is based on constraint satisfaction, namely temporal consistency techniques are used (Planken 2008). Note that we start with an existing schedule, where some constraints are violated. We first relax the assignment of temporal variables, and simply by temporal propagation we try to find a feasible schedule. This gives as temporal windows showing where particular activities can be allocated. We then try to allocate each activity to its original time, if the time is inside the corresponding time window (otherwise, the boundary value of the interval which is closest to the original time is used). This may violate some constraints, so we try to shift the activities in time to repair locally these constraints. This second process of activity shifting is identical to the algorithm in (Barták and Skalický 2009) but now the temporal constraints are assumed.

Analyzer

Finally, the *Analyzer* uses the ideas of critical path to discover weak parts of the schedule. Briefly speaking, we find the activities that are responsible for delay of some process. For these activities, we try to decrease the delay, which can be done by shortening the duration of the activity or by adding a new resource where the activity can be processed. Currently, we use ad-hoc rules to suggest the improvements (overloaded resource → add a new resource). The improvements are then applied to the scheduling model and the Optimizer generates a new schedule, whose cost is used to evaluate the improvement. Some interactions between the possible improvements are also discovered during this process. For example, we can find that if two improvements are applied together then their effect is annihilated. From the set of possible improvements, a subset with the best overall cost is selected by using the techniques of *project portfolio optimization*. Again, the problem is modeled as a constraint satisfaction problem and ILOG CP Optimizer is used to solve it. Briefly speaking Boolean variables are used to describe whether an improvement is selected and constraints describe the relations between the improvements found during the evaluation stage. A standard branch-and-bound procedure is used to do optimization.

Conclusions

The FlowOpt system is intended to demonstrate that non-experts in optimization can use advanced optimization technology. Hence a lot of effort was put in the design of an intuitive user interface. One of its design principles is hiding the formal modeling and algorithmic complexity from the user while preserving the full flexibility. The most complex algorithms such as workflow verification, scheduling, and schedule repair are completely hidden from the user – we call it a "press-button" approach. The user presses a button in GUI and the system fully automatically solves the problem. However, from our experience the users do not like black-box solutions and they require a full control of the system. This is exactly what we offer in the FlowOpt by allowing the users to manually influence every step of the production planning process starting with the design of workflows and ending with the possibility to modify the generated production plans. Hence despite automating many routine tasks, the users still keep control of the system. In summary the most innovative features of the FlowOpt system includes:

- an editor of structured workflows with fully automated verification,
- a generic scheduler based on constraint model that is generated automatically from the problem specification
- an interactive Gantt Viewer connected to the problem specification and supporting manual plan modification with automated correction of violated constraints
- a novel concept of suggesting improvements of the production environment based on schedule analysis.

The FlowOpt system is meant as a demonstration of novel user-friendly approach to production scheduling. For applicability in real life environments, it requires some extensions especially in the modeling of resources. Though unary resources are probably the most frequent in practice it is still important to support resources with capacity greater than one. Also, modeling availability of resources via calendars seems very important for customers as well as modeling interruptible activities. To support these features, the Optimizer needs to be modified and in part also the Gantt Viewer. The schedule repair is currently realized via shifting activities in time, but there are other opportunities like re-allocating them to a different resource or event trying another alternative from the workflow. The current

Analyzer shows the first steps in automated analysis of the enterprise and generating suggestions for improvement of the production facility. This is still an open research area with many opportunities. The final major challenge is on the other side of the planning process – automated construction of structured workflows from the description of primitive operations (knowledge engineering).

Acknowledgments. The research and development is supported by the Czech Science Foundation under the contract P202/10/1188 and by EU Funding Scheme Research for the benefit of SMEs: FP7-SME-2007-1 under the project ValuePOLE (contract 222218). We thank the reviewers for valuable comments and suggestions.

References

Bae, J.; Bae, H.; Kang, S.-H.; Kim, Z.; 2004. Automatic Control of Workflow Processes Using ECA Rules. *IEEE Transactions on Knowledge and Data Engineering*, **16**(8), 1010-1023.

Barták, R.; 2012. On Complexity of Verifying Nested Workflows with Extra Constraints. *Proceedings of 4th International Conference on Agents and Artificial Intelligence (ICAART 2012),* Volume 1, pp. 346-354, SciTePress.

Barták, R. and Čepek, O.; 2008. Nested Precedence Networks with Alternatives: Recognition, Tractability, and Models. In *Artificial Intelligence: Methodology, Systems, and Applications* (AIMSA 2008). LNAI 5253, Springer Verlag, pp. 235-246.

Barták, R.; Little, J.; Manzano, O.; Sheahan, C.; 2010. From Enterprise Models to Scheduling Models: Bridging the Gap. *Journal of Intelligent Manufacturing*, **21**(1), 121-132, Springer Verlag.

Barták, R.; Sheahan C.; Sheahan, A.; 2012. MAK€ – A System for Modelling, Optimising, and Analyzing Production in Small and Medium Enterprises. In *Proceedings of 38th International Conference on Current Trends in Theory and Practice of Computer Science* (SOFSEM). LNCS 7147, Springer Verlag, pp. 600-611,

Barták, R. and Skalický, T.; 2009. A local approach to automated correction of violated precedence and resource constraints in manually altered schedules. In *Proceedings of MISTA 2009: Fourth Multidisciplinary International Scheduling Conference: Theory and Applications*, Dublin, Ireland, pp. 507-517.

Beck, J. Ch. and Fox, M.S.; 2000. Constraint-directed techniques for scheduling alternative activities. *Artificial Intelligence,* **121**, 211-250.

Bi, H. H.; Zhao, J. L.; 2004. Applying Propositional Logic to Workflow Verification. *Information Technology and Management*, **5**(3-4), 293-318.

Do, M.; Okajima, K.; Uckun, S.; Hasegawa, F.; Kawano, Y.; Tanaka, K.; Crawford, L.; Zhang, Y.; Ohashi, A.; 2011. Online Planning for a Material Control System for Liquid Crystal Display Manufacturing. *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling* (ICAPS) AAAI Press, pp. 50-57.

Giro, S.; 2007. Workflow Verification: A New Tower of Babel. In *AIS-CMS International Modeling and Simulation Multiconference*, Buenos Aires, Argentina.

González-Ferrer, A.; Fernández-Olivares, J.; Castillo, L.; 2009. JABBAH: A Java Application Framework for the Translation Between Business Process Models and HTN. *Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling* (ICKEPS). Thessaloniki, Greece, pp. 28-37.

Kim, P.; Williams, B.; Abramson, M.; 2001. Executing Reactive, Model-based Programs through Graph-based Temporal Planning. In *Proceedings of International Joint Conference on Artificial Intelligence* (IJCAI), pp. 487-493.

Kuster, J.; Jannach, D.; Friedrich, G.; 2007. Handling Alternative Activities in Resource-Constrained Project Scheduling Problems. In *Proceedings of Twentieth International Joint Conference on Artificial Intelligence* (IJCAI-07), pp. 1960-1965.

Laborie, P.; 2009. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (CP-AI-OR), LNCS 5546, Springer Verlag, pp. 148-162.

Laborie, P.; Rogerie, J.; 2008. Reasoning with Conditional Time-intervals. In *Proceedings of the Twenty-First International Florida AI Research Society Conference* (FLAIRS), AAAI Press, 2008, pp. 555-560.

Planken, L.R.; 2008. *New Algorithms for the Simple Temporal Problem*. Master Thesis, Delft University ofTechnology.

Rovenský, V.; 2011. *Workflow Modelling*. Master Thesis, Charles University in Prague.

van der Aalst, W.; Hofstede, A. H. M. t.; 2005. Yawl: Yet another workflow language. *Information Systems*, **30**(4), 245-275.