Planning and Rescheduling

Marián Lekavý[*]
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
lekavy@fiit.stuba.sk

Abstract
This thesis presents results in two areas. The first area is automated planning. Approaches to the domain-independent automated planning can be divided into two groups: STRIPS-like planning (based on operators) and HTN-like planning (based on hierarchical decomposition). It is widely believed, that the expressivity of STRIPS-like planning is lower than the expressivity of HTN-like planning. This would mean that HTN-like planning can solve more problems than STRIPS-like planning. In this thesis we show, that the expressivity of both approaches is identical and that both approaches can solve all problems solvable by a finite tape Turing machine (i.e. solvable by a common computer). The second part of this thesis addresses rescheduling. If we have a schedule with unmovable deadline, we need to react on late activities by shortening of the remaining schedule part. In this thesis, we show that every rescheduling corresponds to a graph cut or the superposition of several graph cuts. Based on this fact, we designed a new algorithm for the cheapest rescheduling. The algorithm is based on the conversion of the rescheduling problem to the problem of minimal graph cut.

Keywords
planning, STRIPS, HTN, expressivity, rescheduling, graph cut, workflow system

1. Introduction
The beginning of symbolical planning is connected to the system STRIPS (STanford Research Institute Problem Solver) [5]. Its main goal was to separate the plan search mechanism from the domain, for which we want to find the plan. The domain has to be written down as a description of initial and final state of the environment and a description of operators that can change the state of the environment. Every operator is described as a triple:

$$operator = (pre, del, adds),$$

where pre, del and $adds$ are sets of literals. The set pre contains the operator precondition, i.e. the literals that have to be valid in the state, in which the operator can be applied. Before an operator is executed, it is necessary to verify that its precondition is fulfilled in the current state of the world. The sets of literals del and $adds$ contain literals, which are added/removed to/from the actual state of the world after the operator is executed. That means, that the modification of the world state is expressed by adding literals from the $adds$ set and removing literals from the del set. The state space, defined this way, is then automatically searched for a route from the initial to the final state.

The second, equally important, approach to symbolical planning is HTN (Hierarchical Task Network). The main idea of this approach was introduced by the authors of STRIPS. This idea was transformed to its present form later, in the ABSTRIPS system [12] (which allowed to define several hierarchically ordered STRIPS domains) and later in the HTN itself [13]. HTN-like planning is based on tasks, which can be decomposed to simpler tasks. The domain definition contains rules of tasks decompositions, called task networks. Formally, a task network can be expressed as:

$$((n_1, \alpha_1), ..., (n_m, \alpha_m), \phi),$$

where α_i is the (sub)task marked by the identifier n_i and ϕ is a formula describing the condition, which has to be fulfilled before the task network can be used.

For a long time, it was assumed, that HTN-like planning can solve more more domains than STRIPS-like planning (i.e. it has larger expressivity). There is even a proof for this assumption [4]. However, this assumption is not correct and in this thesis we show, that the original proof contains inaccuracies and that in contrast to the mainstream opinion, STRIPS-like planning (and even the original STRIPS) is able to solve the same class of problems as HTN-like planning. We also show, that the expressivity of both mentioned planning approaches is identical to the expressivity of a finite-tape Turing machine. Therefore, with both planning approaches (including the original STRIPS, which is more than 35 years old), we can solve all problems solvable by a common computer.

[*]Recommended by thesis supervisor: Prof. Pavol Návrat Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on March 25, 2009.

Planning is an activity, which determines what steps we need to execute to reach a goal and in what order we need to execute these steps. Based on the plan, it is usually necessary to create a schedule of plan execution, i.e. to determine the execution time of individual plan steps. During plan execution, different unforeseen events may occur, affecting and changing the plan execution. The schedule needs to be adjusted to these changes.

There are two different groups of approaches to schedule creation: creating new schedule from scratch if there was no previous schedule (e.g. MicroBoss [14]); and rescheduling (schedule adjustment) if the original schedule became inaccurate and it is necessary to change it. It is possible to use the first group of approaches (creating new schedule from scratch) to adjust a schedule, but by doing it, we loose information, which was contained in the original schedule. Moreover, it is possible, that the number of changes in the schedule will be higher than necessary. Therefore, the rescheduling approaches are very important.

Sometimes we need to adjust a schedule without changing the total time of schedule execution (makespan). It is the case of projects, where the schedule end is contracted and unchangeable. If the end time of the schedule is unmovable, we can use 3 basic approaches. The most common approach is to use moving of activities in combination with robust schedule (for example Match-Up Rescheduling [3, 1]). Reserves, inserted between activities, can absorb possible activities lateness. We can also transfer part of the activities to a different organisation (subcontracting), for example [2]. This is, in fact, increasing of parallelism. The third possibility is to allow shortening of activities, in addition to moving of activities.

Automated approaches to schedule adjustment are usually designed for manufacturing processes control or parallel computations control, where individual activities cannot be shortened. This is the reason, why most rescheduling approaches don't allow shortening of activities as a valid solution. On the other hand, in the domain of project management, where individual activities are executed by humans, shortening or even cancelling of activities is used very often. However, approaches to rescheduling in the domain of project management are usually not automated and focus mainly on supporting a human manager, who adjusts the schedule manually.

This thesis focuses on automated schedule adjusting in the domain of project management, where shortening of activities is an important instrument used to assure deadlines fulfilment. This thesis shows that the cheapest schedule shortening corresponds to minimal graph cut in the dependencies graph of the schedule. The thesis also proposes a new approach to schedule shortening based on this fact.

2. Thesis Goals

This thesis follows goals in two areas: in the area of planning and in the area of rescheduling.

In the area of planning, the main goal is to investigate the expressivity of two basic symbolical planning approaches: STRIPS-like planning and HTN-like planning. In this context, expressivity is defined by domains, which can be written down and solved by some approach. The main goal is to compare the expressivity of these two approaches, but also to delimit the expressivity of both approaches in the context of other computational systems.

In the area of rescheduling, the main goal is to show the relationship between schedule change and graph cut. Additionally, the thesis focuses on the design of a new approach to schedule shortening based on graph cut. The main goal of this approach is to allow automatical shortening and moving of activities, which assures the final deadline fulfilment even if some activities fail to end in time.

3. STRIPS-Like and HTN-Like Planning Expressivity

There are two meanings of the term "expressivity" in the area of automated planning. In the first meaning, it is the set of problems (or the size of this set), which can be expressed and solved by some system. In the case of planning, it is the set of plans, which can be created by some system. The second meaning of the term "expressivity" describes, how simple/complicated is to describe a problem or any other information in some system. In this case, expressivity is usually subjective, as it is connected to the way, how a human can express and write down his/hers thoughts.

The first meaning describes, *what* we can write down and solve in a given (planning) system, while the second meaning describes, *what efford* we need to do it. This thesis focuses on the first meaning of expressivity only, i.e. it focuses on the question, what problems we can express and solve with the help of STRIPS-like planning and HTN-like planning. Generally, it is assumed, that the expressivity of HTN-like planning is larger than the expressivity of HTN-like planning. There are two proofs, which show, that HTN-like planning is more expressive [4].

The first proof is based on transforming the planning problem to grammars. Subsequently, it claims, that the STRIPS-like planning corresponds to regular grammars, while HTN-like planning corresponds to context free grammars. As an implication of this, HTN-like planning covers a larger class of domains. This proof looks very simple and comprehensible and therefore it is very popular in the planning community.

However, in the case of STRIPS, the transformation to a regular grammar is not correct. The STRIPS states (i.e. states of the world) are not atomic, but composed of literals. It is possible to represent a state as a set of literals ($S = \{L_1, L_2, L_3, ...\}$). The grammar rules for STRIPS ($S_0 \rightarrow a_{01}S_1$) are not atomic either and can be further decomposed into: $S_0 = \{L_{P_1}, L_{P_2}, ...\} \cup \{L_1, L_2, ...\} \rightarrow a_{01}(S_0 \setminus \{L_{D_1}, L_{D_2}, ...\}) \cup \{L_{A_1}, L_{A_2}, ...\}$, where $\{L_{P_1}, L_{P_2}, ...\}$, $\{L_{D_1}, L_{D_2}, ...\}$ and $\{L_{A_1}, L_{A_2}, ...\}$ are sets of literals forming the precondition and effects (added and removed literals) of the action a_{01}.

Therefore, plan derivation in STRIPS is not just a sequential transformation of atomic states. Contrary, it is a transformation of composite states according to rules, which cannot be expressed by a regular grammar.

The second proof shows, that the theoretical model of HTN is undecidable, while STRIPS and STRIPS-like planning are decidable problems. The main reason causing

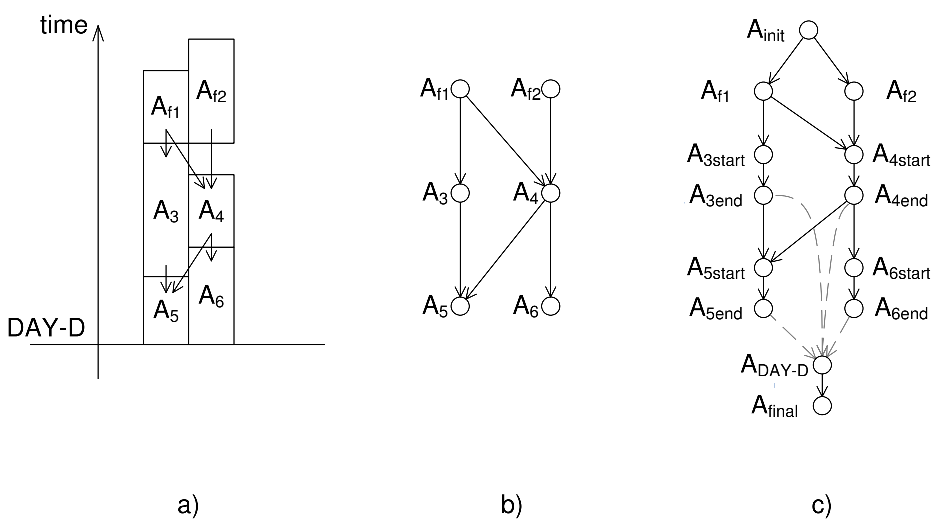


Figure 1: Simple example of schedule conversion from activities and dependencies (a) to oriented graph (b) and splitting of activities vertices and adding of the final deadline, artificial source and sink (c).

this result is the fact that the theoretical HTN model used in the proof uses an infinite set of symbols to mark tasks, so it is able to create an infinite space of plans.

The theoretical model of HTN is, however, not practically usable, because of its undecidability. This is true even with very strong restrictions of the model. Therefore, practically used planners use different modifications, which restrict the space of plans to be finite and change the problem to be decidable [4]. The most common restrictions of HTN are:

- Restricting the plan length. If the plan length is finite, the space of all possible plans is finite too, as we choose of finite number of actions while adding an action to a plan.
- Restricting the tasks to be acyclic. It is not allowed to decompose any task into itself (after several steps of decomposition). This way, every task can only be decomposed up to a finite depth, which is smaller than the total number of tasks.
- Restricting the task networks to be fully ordered. Tasks are fulfilled sequentially one after another, so subtasks of individual tasks cannot overlap in time.

Any of these restrictions is sufficient to make HTN-like planning decidable and therefore practically usable. All planners based on HTN use at least one of these restrictions (or their slight modifications). This is the reason, why it is better to use the term “HTN-like planning” for planners based on the HTN model and using one of these restrictions, than using it for the theoretical HTN model without restrictions. Based on this definition of HTN-like planning, it is possible to formulate the following theorem [8]:

THEOREM 1. *Every HTN-like domain can be expressed as a STRIPS-like domain. Every STRIPS-like domain can be expressed as a HTN-like domain. Therefore, the expressivity of STRIPS-like planning is identical to the expressivity of HTN-like planning.*

PROOF SKETCH 1. *The plan space of a HTN-like domain is finite. With the help of finite number of plans, we can reach a finite number of final states, starting from one initial state. This means, that the state space of this domain is finite. For a finite state-space, we can construct a STRIPS-like domain by simply enumerating all possible state transitions as STRIPS actions. As a result, STRIPS-like planning expressivity is not smaller than the expressivity of HTN-like planning. The second half of the proof, showing that HTN-like planning expressivity is not smaller than the expressivity of STRIPS-like planning, is constructive, can be found in [4] and is based on the transformation of STRIPS-like domain to a flat HTN-like domain, where it is possible to decompose the initial task into any primitive task (action) and the relationships between actions can be expressed by additional restrictions of primitive tasks*

Enumerating all states in a domain is not very practical and can lead to exponential number of actions. In the thesis, a conversion of HTN-like domains to STRIPS-like domains is shown, which can be made in polynomial time and which does not change the time complexity of plan search. The conversion is based on the idea of using the STRIPS planner as an emulator of the HTN-like planner. The resulting STRIPS plan corresponds to the sequence of steps, which the HTN-like planner has to execute.

Additionally, the thesis proposes a conversion of a finite-tape Turing machine to a STRIPS domain. The conversion is based on using the STRIPS planner as an emulator of the finite-tape Turing machine. The resulting plan consists of steps, which the Turing machine has to execute in order to halt. This conversion shows that all problems solvable by a finite-tape Turing machine (and therefore also solvable by a common computer) can be solved by a STRIPS-like planner or a HTN-like planner.

4. Rescheduling and Minimal Graph Cut

Dependencies between individual activities can be represented by documents. If some activity A_{use} uses the document D created by some different activity $A_{produce}$, then the activity A_{use} is dependent on the activity $A_{produce}$.

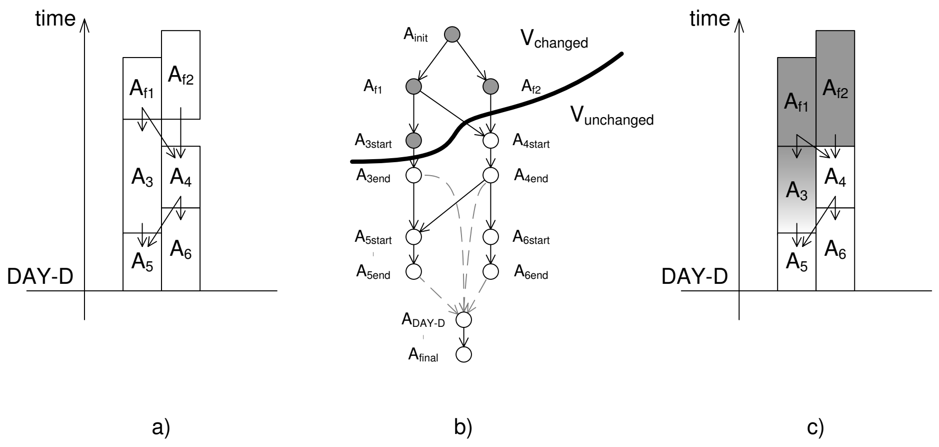


Figure 2: Simple example of rescheduling based on graph cut. From the initial schedule (a) we create a dependencies graph and according to a minimal graph cut (b) we compensate the lateness of the activities A_{f1} a A_{f2} by shortening of the activities activity A_3 (c).

Execution of the activity A_{use} cannot start before the activity $A_{produce}$ ends and produces the document D. If some activities have a dependence between them, but this dependence is not the result of producing and using of documents, we can add an empty document, which will represent this activity.

From the initial schedule and the information about documents transferred between activities (Figure 1a), it is possible to create a dependencies graph. In this graph, activities are represented by vertices, while dependencies are represented by edges. This way, we get an acyclic oriented dependencies graph (Figure 1b). For the needs of rescheduling, we can remove all activities (and adjacent dependencies), which are not dependent on activities $A_{f\#}$. $A_{f\#}$ are the activities, which failed to finish in time (# stands for the identifier of a failed activity).

If we want to allow shortening of activities, we need to have edges for individual activities too. Therefore, we divide every vertex of the dependencies graph $A_{\#}$, representing some activity except the activities $A_{f\#}$, into two vertices: $A_{\#start}$ and $A_{\#end}$. All edges, originally ending in $A_{\#}$, will now end in $A_{\#start}$ and all edges, originally starting from $A_{\#}$, will now start from $A_{\#end}$. At the same time, we add an edge $e_{A\#}$ leading from $A_{\#start}$ to $A_{\#end}$.

Additionally, we add the edge A_{DAY-D}, representing the final deadline of the schedule (Day D). A_{DAY-D} is dependent on every activity in the schedule, as all activities need to be finished before the final deadline. Finally, we add an artificial source A_{init} and sink A_{final}. We connect the source A_{init} to all failed activities $A_{f\#}$. The final deadline A_{DAY-D} will be connected to the artificial sink A_{final}. An example of the resulting dependencies graph is in Figure 1c.

This way, we created a dependencies graph, where vertices represent events in the schedule, which may be affected by the change of duration of the failed activities $A_{f\#}$. These events are the ends of the failed activities ($A_{f\#}$), starts ($A_{\#start}$) and ends ($A_{\#end}$) of dependent activities and the final deadline (A_{DAY-D}).

For a dependencies graph created this way, we can formulate the following theorem [9]:

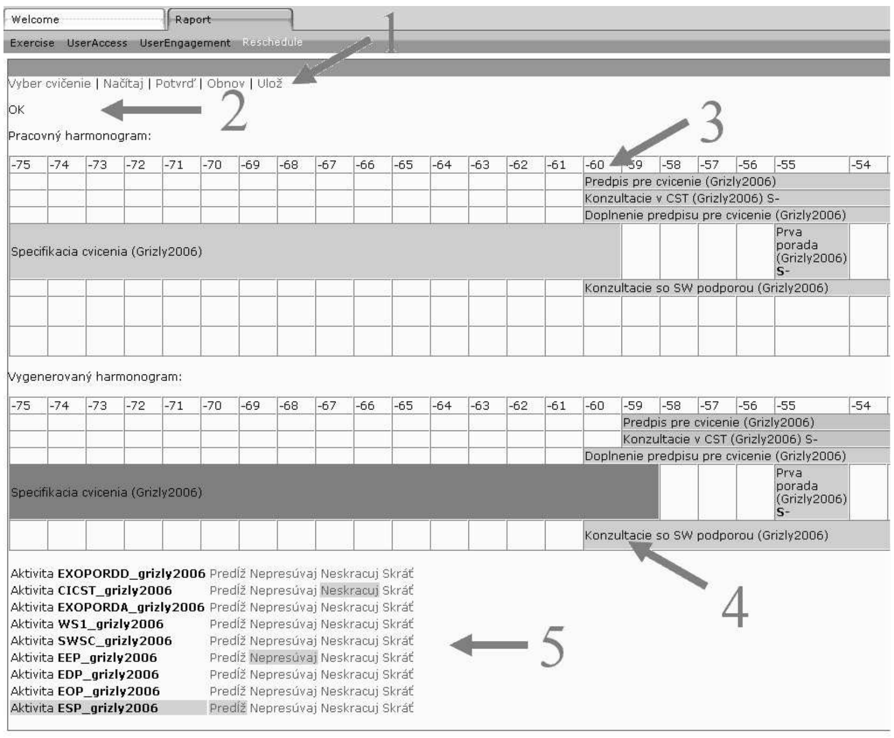
Theorem 2. *Every possible rescheduling, which changes the schedule by postponing the start and/or end times of some activities by 1 time unit, corresponds to some cut of the dependency graph.*

Proof sketch 2. *We define two sets: $V_{changed}$ containing vertices which are postponed in the rescheduling and $V_{unchanged}$ containing vertices which are not affected by the rescheduling. Each event of the dependency graph ($A_{\#start}$, $A_{\#end}$, A_{DAY-D}, $A_{f\#}$; # standing for an activity identifier) is either postponed or not, i.e. it belongs to $V_{changed}$ or $V_{unchanged}$, while $V_{changed} \cap V_{unchanged} = \emptyset$, as no event can be postponed and not postponed at the same time. A graph cut is the division of a graph into two sets, dividing two vertices (in this case $A_{init} \in V_{changed}$ and $A_{final} \in V_{unchanged}$). This means that every possible combination of sets ($V_{changed}, V_{unchanged}$) representing some rescheduling is a cut of the dependency graph. At the same time, for every graph cut of the dependencies graph ($V_{changed}, V_{unchanged}$) there is a corresponding rescheduling.*

Similarly, it is possible to show, that every rescheduling (changing the beginnings and/or ends of activities by arbitrary time) corresponds to a superposition of several graph cuts. The proof can be found in the thesis.

With correctly defined costs of edges in the dependencies graph, it is possible to find the cheapest rescheduling by finding the minimal cut in the dependencies graph. The meanings of individual edges costs are:

- Shortening of an activity:
  Cost of the edge $e_{A\#} = (A_{\#start}, A_{\#end})$.
- Shortening of a dependency:
  Cost of the edge $e_{A_iA_j} = (A_{iend}, A_{jstart})$ (or $e_{A_{fi}A_j} = (A_{fi}, A_{jstart})$ for the failed activities).
- Moving of an activity:
  Cost of the edge $e_{A_\#D} = (A_{end}, A_{DAY-D})$.
- Deadline violation:
  Cost of the edge $e_{DA_{final}} = (A_{DAY-D}, A_{final})$.



OK

Pracovný harmonogram:

-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54
															Predpis pre cvicenie (Grizly2006)						
															Konzultacie v CST (Grizly2006) S-						
															Doplnenie predpisu pre cvicenie (Grizly2006)						
Specifikacia cvicenia (Grizly2006)																				Prva porada (Grizly2006) **S-**	
															Konzultacie so SW podporou (Grizly2006)						

Vygenerovaný harmonogram:

-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54
																Predpis pre cvicenie (Grizly2006)					
																Konzultacie v CST (Grizly2006) S-					
															Doplnenie predpisu pre cvicenie (Grizly2006)						
Specifikacia cvicenia (Grizly2006)																				Prva porada (Grizly2006) **S-**	
															Konzultacie so SW podporou (Grizly2006)						

Aktivita **EXOPORDD_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **CICST_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **EXOPORDA_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **WS1_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **SWSC_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **EEP_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **EDP_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **EOP_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť
Aktivita **ESP_grizly2006** Predĺž Nepresúvaj Neskracuj Skráť

Figure 3: Portlet for rescheduling in the RAPORT portal. 1. Menu, 2. Status line, 3. Working schedule, 4. Generated schedule, 5. List of activities.

- Aborting a failed activity:
  Cost of the edge $e_{A_{init}A_{f\#}} = (A_{DAY-D}, A_{f\#})$.

Edges between vertices inside the same set $V_{changed}$ or $V_{unchanged}$ correspond to activities or dependencies with unchanged length, because either both, the starting and ending vertex, are moved (in the case $V_{changed}$) or neither of the two vertices is moved ($V_{unchanged}$). Activities and dependencies with edges going from $V_{changed}$ to $V_{unchanged}$ are shortened by 1 time unit, because their starting vertex is moved, while their ending vertex remains unchanged (Figure 2).

The proposed approach to rescheduling according to minimal graph cut was employed in the RAPORT project, for which it was primarily designed. The RAPORT system was created in the cooperation of the Faculty of Informatics and Information Technologies of the Slovak University of Technology in Bratislava, the Institute of Informatics of the Slovak Academy of Sciences and the National Academy of Defence. The RAPORT system [6, 11] was designed for a pilot application: organisation of military exercises in the Centre of Simulation Technologies (CST) of the National Academy of Defence (NAO) in Liptovský Mikuláš.

An important attribute of military exercises organisation in CST NAO is, that at the day of the exercise (Day D), all activities have to be successfully accomplished in order to assure success of the exercise. If any activity is late, the remaining schedule has to be shortened to avoid the violation of the exercise time (time of the schedule end).

The approach to rescheduling by using minimal graph cut was implemented in the RAPORT system as a separate portlet (Figure 3). This portlet allows semiautomatic schedule adjustment: the system offers a solution and the user can confirm this solution, but the user is also allowed to further modify the offered solution.

The algorithm of rescheduling by using minimal graph cut offers several possibilities of further research and improvement. It will be very useful to combine this method with existing scheduling methods, especially with approaches based on CPM and PERT[10]. Another possibility is to allow restrictions, resulting from using resources by individual activities, for example in the form of resource links, which are a part of the critical chain method [7].

5. Thesis Contributions

This thesis has contributions in two areas:

- Determining the expressivity of STRIPS-like and HTN-like planning:
  - Expressivity of STRIPS-like planning and HTN-like planning is identical. This means, that all problems solvable by one of these approaches are also solvable by the second approach. At the same time, this thesis invalidates the generally accepted opinion, that the expressivity of approaches based on HTN is larger than the expressivity of approaches based on STRIPS.
  - HTN-like domains can be converted to equivalent STRIPS domains in polynomial time. The conversion in the opposite direction was already known earlier.
  - The expressivity of both planning approaches is identical to the expressivity of a finite tape Turing machine. This means, that STRIPS-like and HTN-like planners can solve all problems solvable by a common computer.

- Creating of an approach to rescheduling by using minimal graph cut:
  - Proof, that every time change of a schedule corresponds to a cat or a superposition of cuts of a dependencies graph.
  - Creating a new approach to rescheduling based on minimal graph cut. The approach is based on the fact that the cheapest change of a schedule corresponds to a minimal cut in a dependencies graph.
  - Implementation and validation of the approach in the workflow system for military exercises preparation RAPORT.

References

[1] M.S. Akturk, E. Gorgulu. Match-up scheduling under a machine breakdown. In *European Journal of Operational Research*, volume 112, pages 81–97, 1999.

[2] H. Arslan, H. Ayhan, T.L. Olsen. Analytic models for when and how to expedite in make-to-order systems. In *IIE Transactions*, volume 33, pages 1019–1029, 2001.

[3] J.C. Bean, J.R. Birge, J. Mittenehal, C. E. Noon. Match-up scheduling with multiple resources, release dates and disruption. *Operations Research*, 39(3), pages 470–483, 1991.

[4] K. Erol, D. Nau, J. Hendler. HTN planning: Complexity and expressivity. In *Proc. of the 12th National Conf. on Artificial Intelligence* (AAAI-94) AAAI Press, Menlo Park, USA, 1994.

[5] R.E. Fikes, N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence 2*, Elsevier Science Publishers Ltd., Essex, UK, pagesv189–208, 1971.

[6] R. Forgac, I. Budinska, E. Gatial, G. Nguyen, M. Laclavik, Z. Balogh, I. Mokris, L. Hluchy, M. Ciglan, M. Babik. Ontology based knowledge management for organizational learning. In *Proc. of 9-th Int. Conf. ISIM'06 – Information Systems Implementation and Modelling*, Brno, MARQ Ostrava, pages 177–184, 2006.

[7] E.M. Goldratt. Critical Chain. Gover Publishing, 1997.

[8] M. Lekavy, P. Návrat. Expressivity of STRIPS-Like and HTN-Like Planning. In *Lecture Notes in Artificial Intelligence*. Subseries of LNCS, Vol. 4496, *Agent and multi-agent Systems. Technologies and applications. 1st KES Int. Symposium, KES-AMSTA 2007*, Wroclaw, Poland, 2007, Springer-Verlag Berlin Heidelberg, pages 121–130, 2007.

[9] M. Lekavy, P. Návrat. Extension of Rescheduling Based on Minimal Graph Cut. In *Lecture Notes in Computer Science (LNCS)*, volume 4910, *SOFSEM 2008: Theory and Practice of Computer Science*, Novy Smokovec, Slovakia, 2008, Springer-Verlag Berlin Heidelberg, pages 340–351, 2008.

[10] J.J Moder, C.R. Phillips, E.W. Davis. Project management with CPM, PERT, and precedence diagramming. Van Nostrand Reinhold Company, NY, 1983.

[11] RAPORT – Research and development of a knowledge based system to support workflow management in organizations with administrative processes (APVT-51-024604). `http://raport.ui.sav.sk/`

[12] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Artificial intelligence 5*. Elsevier Science Publishers Ltd., Essex, UK, 1974.

[13] E.D. Sacerdoti. A Structure for Plans and Behavior, Elsevier-North Holland, 1977.

[14] N. Sadeh. Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling, *Ph.D. Thesis*. School of Computer Science, Carnegie Mellon University, 1991.

Selected Papers by the Author

M. Lekavý. Multiagent systems. In M. Bieliková, P. Návrat, editors, *Selected Studies on Software and Information Systems 1*. Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava, 2006. (in Slovak)

M. Lekavý, P. Návrat. Expressivity of STRIPS-Like and HTN-Like Planning. In *Agent and multi-agent Systems. Technologies and applications – KES-AMSTA 2007*, LNCS 4496, Wroclaw, Poland, 2007, Springer, pages 121–130, 2007.

M. Lekavý, P. Návrat. Extension of Rescheduling Based on Minimal Graph Cut. In *SOFSEM 2008: Theory and Practice of Computer Science*, LNCS 4910, Nový Smokovec, Slovakia, 2008, Springer, pages 340–351, 2008.

M. Lekavý, P. Návrat. Dynamic Rescheduling as a Minimal Graph Cut Problem. In *Software Engineering in Progress – Proc. from work-in-progress of 2nd IFIP Central and East European Conf. on Software Engineering Techniques CEE-SET 2007*, Poznan, Poland, 2007, pages 141–153, 2007.

M. Lekavý, J. Wagner. Various Uses of Potential Map in a Soccer Game. In *ZNALOSTI 2008*, Bratislava, Slovakia, 2008, STU, Bratislava, Slovakia, pages 112–123, 2008.

M. Lekavý: Quantum Random Walks Without a Quantum Computer. In M. Bieliková, editor, *Proc. in Informatics and Information Technologies Student Research Conf. 2008*, 2008, Bratislava, Slovakia, 2008, pages 319–326, 2008.

M. Lekavý: Approaches to RoboCup Soccer Simulation. In M. Bieliková, editor, *Proc. in Informatics and Information Technologies Student Research Conf. 2007*, April 18 2007, Bratislava, Slovakia, pages 51–58, 2007.

M. Lekavý, P. Návrat. Dynamic Workflow Schedule Adjusting. In *INFORMATICS 2007, Proc. of the 9th Int. Conf. on Informatics*. Bratislava SSAKI, pages 117–123, 2007.

M. Lekavý, P. Návrat. Dynamic Time Table Change Using Minimal Graph Cut. In *Information Technologies – Application and Theory, Proc. from ITAT 2007*, Poľana, pages 9–14, 2007. (in Slovak)

M. Lekavý, P. Návrat. Expressivity of Planning Based On STRIPS and HTN. In *Information Technologies – Applications and Theory, Proc. from ITAT 2007*, Poľana, pages 83–88, 2007. (in Slovak)

M. Lekavý: Multi-Action Backchaining – an algorithm for creating parallel plans. In M. Bieliková, editor, *Proc. in Informatics and Information Technologies Student Research Conf. 2006*, Bratislava, Slovakia, pages 212–219, 2006.

M. Lekavý, P. Návrat. Contingent Parallel Plans Based On Operators. In *Proc. from ITAT 2006*, 2006, Nízke Tatry, pages 113–117, 2006.

M. Lekavý, P. Návrat. Team cooperation in a simulated real-world environment – shared plans in a simulated soccer game. In *ZNALOSTI 2005*, Poster proceedings, pages 69–72, 2005.

M. Lekavý, P. Návrat, R. Kapusta, M. Ulický, M. Koštál, M. Košík, M. Multiagent coordination in the domain of robotic soccer. In *ZNALOSTI 2005*, Poster proceedings, pages 65–68, 2005.

M. Lekavý: Optimising multi-agent cooperation using evolutionary algorithm. In M. Bieliková, editor, *Proc. of Informatics and Information Technologies Student Research Conference*, Bratislava, Slovakia, pages 49–56, 2005.