# Grid Workflow Recovery as Dynamic Constraint Satisfaction Problem

Stanimir Dragiev, Joerg Schneider
*Department of Electrical Engineering and Computer Sciences*
*Technische Universitaet Berlin*
*Berlin, Germany*
{*stanio,komm*}*@cs.tu-berlin.de*

***Abstract*—With service level agreements (SLAs) the Grid broker guarantees to finish the Grid jobs by a given deadline. There are a number of approaches, to plan reservations to fulfil these deadline requirements and to handle currently running jobs in the case of a resource failure. However, there is a lack of strategies to handle the already planned but not yet started jobs. These jobs will be most likely also affected by the resource failure and can be remapped to other resources well in advance. Complex Grid jobs (Grid workflows) consisting of multiple sub-jobs introduce a higher complexity to determine a remapping saving as much Grid jobs as possible. In this paper a recovery scheme for Grid workflows using a dynamic constraint solver is presented and the gain in the number of saved Grid jobs is evaluated using extensive simulations.**

## I. Introduction

Grid computing opens lots of new perspectives for the application of the traditional information technologies. They range from on-demand high performance computational power for home usage to realisation of complex scientific applications involving multiple enterprises from different cultures, background, locations. At the same time, the new usage possibilities and the nature of the grid give rise to new challenges. We adopt a common view of the grid as a physically wide distributed collection of resources of different types which are spread across administrative and state boundaries.

The idea to reproduce the organisational structure of the resources is the basis for the design [1] of the Virtual Resource Manager (VRM). It consists roughly of two layers: An administrative domain controller (ADC) is concerned with co-allocation and other grid-specific functions and allows for nested administrative domains (AD); Every managed resource is registered with the ADC via an active interface (AI) which addresses the heterogeneity by implementing the capabilities needed for participation in the grid and thus giving a unified view for all resources.

The central question for the ADC is how to provide reliable information on the ability to process given reservation request, as well as to give QoS guarantees and to adhere to the negotiated service level agreements (SLA). To this end, the architecture of the ADC encompasses three stages of scheduling: Triggered by an incoming request for reservation, an online scheduler uses heuristics to produce a valid schedule which meets the needs of the reservation, without breaking already accepted SLAs. As a next stage, a background scheduler optimises the effective schedule towards the policy of the administrative entity. The third stage is concerned with the handling of resource shortcomings resulting from failures.

Dealing with failures has two aspects: support for jobs directly affected by a resource crash and support for jobs scheduled to start on the failed resource in the future. The first domain includes techniques like checkpointing, migration and restart, and enjoy much attention from researchers. However, our work is concerned with the second aspect of recovery scheduling, i.e., the remapping of all planned, but not yet started jobs affected by the outage. A fundamental problem in this domain is the uncertainty about the downtime of a given resource. In general, there are no reliable means to determine the nature of a failure at the time it happens; we can merely observe the result of it: the non-availability of the resource. Under- or overestimation of the downtime leads to poor quality of the future schedules and thus to limited satisfaction of the administrative policy [2]. Hence, mechanisms are needed to adapt the reactions to the downtime.

Recovery means, after all, to choose a valid schedule amongst large number of possible ones, whereby the choice is driven by feasibility criteria, job requirements and resource management policies. The domain concerned with precisely this kind of search problems is Constraint programming. Here, we propose to restate the recovery scheduling in terms of Constraint satisfaction problem (CSP) and benefit from approaches used in similar situations in other research areas. We are particularly interested in the concept of CSP with changing constraints over time, Dynamic CSP (DCSP), and the idea of maintaining similar consecutive solutions.

Grid applications usually comprise several phases – data collection, processing, dissemination of results. These phases are realised by sub-jobs, which interact with each other. In other words, there are temporal and spatial dependencies between the single jobs, as well as QoS requirements on the resources responsible for running the jobs. The notion of Grid workflow [3] helps to consider all parts of a large application as a whole and to model the requirements in a

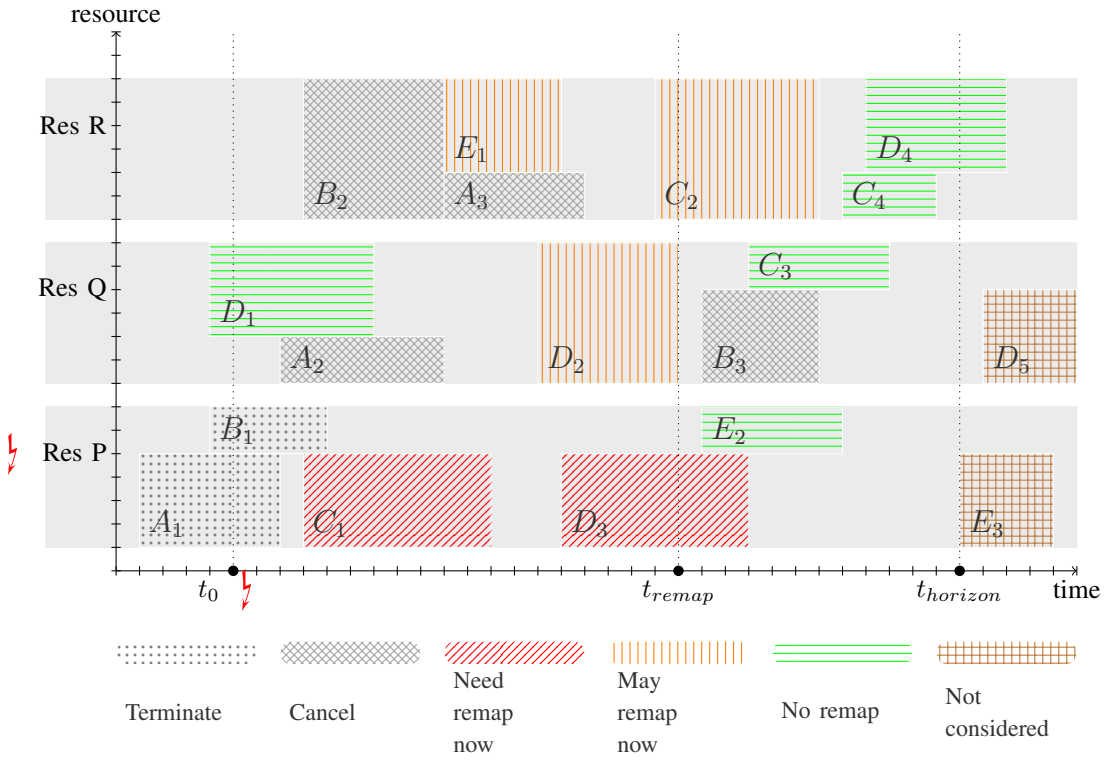


Figure 1. The remapping interval approach for failure recovery extended to support workflows and to limit the search space size. Workflows: $A, B, C, D, E$. Resources: $P$, $Q$ and $R$; Resource $P$ crashes in $t_0$. The approach extension cancels the jobs depending on terminated ones, and doesn't consider the ones beyond $t_{horizon}$.

consistent way. Important property of Grid workflows is that all sub-jobs contribute an essential part to the accomplishment of the final goal of the workflow, i.e. a workflow is successfully completed only if no jobs fail.

In the following section, we comment known efforts to deal with scheduling and recovery in the grid, as well as CSP techniques for similar problems. After discussing our approach to determine a recovery schedule, we show the evaluation results.

## II. Related Work

The simple and robust approaches of managing resources, like providing only batch processing, are still widely used in practice, including MOAB [4], LSF [5], PBS [6], LoadLeveler [7], Globus Toolkit [8]. Despite their maturity, these techniques are not fully capable to ensure the QoS negotiated in service level agreements. The required advance is provided by middleware solutions incorporated in grid management systems like VRM [1] or by advanced local scheduler like CCS[9] or MAUI[10].

In [11] requirements for failure handling in the grid are defined. The emphasis is on the need of application context dependent recovery and distinct recovery on task and workflow level. The task level related techniques include *checkpointing*, *migration*, *restart*, and *replication*. All consideration are concerned with directly affected jobs.

The automatic recovery of future jobs affected by a failure is the topic of [12]. The authors propose an *adaptive*, *load based* downtime independent algorithm. The basic idea behind it is the introduction of a *remapping interval* which is calculated based on the current load situation. The resource is then assumed unavailable for this interval only and all jobs scheduled on the resource for this time are remapped. At the end of the interval the same procedure is repeated until the resource is up again.

The remapping interval computation is based on the average incoming load per timeslot and currently booked load for every particular timeslot. Taken at a particular timeslot, their sum serves as estimation of the expected load in it. The remapping interval is determined as the timeslot for which the expected load falls below given level. This approach handles only independent jobs and is extended in this paper for Grid workflows.

The well studied theory of Constraint satisfaction problems (CSP) can provide means for solving the problem of finding a valid schedule. However, a single CSP depicts the situation at a particular time. When resource availability changes, the constraints change, and this can be seen as new CSP instance. For handling such problems the notion of Dynamic constraint satisfaction problem was introduced

| | |
|---|---|
| Resource capacity | $\forall t \forall r (\sum_{j=0}^{j} s_{t,r,j} \leq R_r.cap(t))$ |
| Workflow atomicity | $\forall j \forall j' (wfl(j,j') \rightarrow (\sum_{t,r} s_{t,r,j} = 0 \rightarrow \sum_{t,r} s_{t,r,j'} = 0))$ |
| Connected assignment | $\forall j \forall r (\forall t_r \forall t_f (rise(t_r, j, r) < fall(t_f, j, r))),$<br>$rise(t_x, r, j) := t_x = -1 \vee (0 \leq t_x < T-1 \wedge s_{t_x,r,j} = 0 \wedge s_{t_x+1,r,j} > 0),$<br>$fall(t_x, r, j) := t_x = T-1 \vee (0 \leq t_x < T-1 \wedge s_{t_x,r,j} > 0 \wedge s_{t_x+1,r,j} = 0).$ |

Figure 2. FOL representation of constraints. Examples.

(DCSP) [13].

The combination of Job shop scheduling problem and Minimal perturbation problem is the topic of [14]. It is based on the idea to maximize similarity between the broken and the new schedule.

One way of maintaining minimal changes in solutions of consecutive CSPs is to produce schedules which are expected to remain valid after changes in the CSP [15]. For this approach to work properly it is assumed that certain types of *expected* changes are known which is limitedly applicable for environment with unexpected resource failures.

An alternative idea is proposed in [16] – to take the previous, usually invalidated by constraint changes CSP solution as basis for the search of a new one. The algorithm works with two subsets of variables – the ones allowed to change and the fixed ones. The approach maximizes the similarity only with respect to the set of fixed ones.

Several general purpose approaches which deal with over-constrained problems are discussed in [17], and this is, in general, the challenge in recovery scheduling. The guide introduces the notions of Fuzzy CSP, Probabilistic CSP, Weighted CSP Partial CSP (PCSP), constraint hierarchies and *higher-order constraints* (in other works referred to as reified constraints) . The latter allow for specifying predicates over constraints, like in $c_i \; or \; true$. They seem to be the best supported ones in existing frameworks, for example, in the constrain solvers built upon Gecode [18] which we employ in our work.

## III. Approach

The recovery scheduling is triggered by a resource failure which possibly makes the existing schedule infeasible. Roughly, the proposed algorithm does the following: determine a *remapping interval*, compute a *horizon* (the maximum considered time for the remapping of jobs), transform the current physical limitations and workflow requirements to constraints, pass the resulting CSP to a constraint solver, implement the found solution as new schedule.

### A. Remapping interval

The computation of the remapping interval is done according to [12]. The adaptation effect of the remapping interval is made dependent on the current load situation. Effectively, it leads to shorter intervals in heavier loads and longer intervals for more relaxed situations. For higher load, this forces the algorithm to eagerly search for possibilities to recover future jobs.

Once the remapping interval is computed it determines the set of jobs we consider for remapping: the ones scheduled to start in the remapping interval – both on broken and intact resources. Figure 1 shows a synthetic schedule and the sorts of job from the recovery scheduler point of view. We consider the not completed jobs on the failed resource terminated. As consequence, their entire workflows are dropped from the schedule (workflows $A$ and $B$ in the example). The jobs scheduled to start in the remapping interval on the failed resource need to be relocated in order to be able to run ($C_1$, $D_3$). By allowing a reordering of other jobs starting in the remapping interval ($D_2$, $E_1$, $C_2$) we gain additional possibilities to obtain optimal schedule.

### B. Schedule Representation

The considered jobs, their dependencies, the timeslots they run on, and the available resources define the space for the search for new schedule. This search space have to be represented in a form suitable for expressing constraints on it. We put it in terms of an integer problem:

*Definition 1:* Let $T := \#\{timeslots\}$, $R := \#\{resources\}$, $J := \#\{jobs\}$. Then, $S_{T \times R \times J}$ is 3D integer matrix, whereas $s_{t,r,j} = x \iff$ "in time slot $t$ job $j$ uses $x$ of resource $r$"; with $s_{t,r,j} \in \{0, J_j.demand\}$

In this representation, we can constrain the variables $s_{t,r,j}$ and employ a constraint solver to find a valid solution.

### C. Constraints

The next task is to elaborate constraints which express the capacity limits of the local resources and the requirements of the accepted reservations. These are basically:

*Resource capacity*: The sum of consumed entities of a resource cannot exceed the resource capacity.

*Enough resources in timeslot*: In every particular timeslot in which a job runs, it has enough resources to run properly.

*Not split*: Every job runs on one resource only.

*Workflow atomicity*: A workflow depends on all its jobs, thus either all jobs can be scheduled, or no job at all. The latter removes needless load form the system.

*Temporal relations*: If job $j_A$ is explicitly specified to run after a $j_B$, the jobs should run in the given order.

*Parallel jobs*: Jobs specified to be parallel need to have resources to run in the same timeslots as long as the longest job requires.

*Enough time*: Jobs have to be assigned to a resource for enough timeslots.

*Types match*: Jobs are assigned to resources capable of running them.

*Data dependencies*: The data prerequisites of a job need to be satisfied, i.e. if job $j_B$ relies on input by job $j_A$, they have to run in order $j_A$ and then $j_B$, whereas if they do not run on the same resource, a transfer job $J_{AB}$ needs to be scheduled between them.

*Connected assignment*: The timeslots a resource is assigned to a job have to be consecutive ones.

*Workflow times*: All jobs start after the start time of the workflow and end before the workflow end time.

The formal definition of the constraints is given in First order logic (FOL). As an example, consider the three constraints of figure 2.

### D. Over-constrained problem

Obviously there are cases, a failure not only invalidates the current schedule but tightens the resource constraints so much that there is no schedule in which all accepted reservations fit. For such over-constrained problems, *reification* allows to relax selected constraints and to find the best available solution. Reification means to break a constraint, in order to make the problem solvable.

This makes sense for constraints which do not represent physical limitations, i.e. only for the ones which determine whether a job runs at all. In our model, it is not feasible to reify, for example *Resource capacity*, or *Workflow times*. Both would lead to a schedule which is infeasible and thus to no benefit from the recovery.

In contrast, the more jobs are dropped from the schedule, the easier to find valid assignment for the rest. In other words, if we allow to break constraints like *Enough resources in timeslot* we can adjust the load to the available resources. Effectively, this ensures the existence of solution and passes the responsibility for determining the optimal schedule to the objective function

### E. Optimisation Objective

The notion of *optimal schedule* implies an objective for the search process. In order to counteract to the reification, we want as the primary goal to maximize the scheduled load[1]: $W(S) = \sum_{t,r,j} s_{t,r,j}$, with $t$ spanning the indexes of the considered timeslots, $r$ – the resource indexes, and $j$ – the job indexes. As a secondary objective we impose on the solution the condition to remain as close as possible to the

[1]this may be seen as a measure for the earnings of the resource provider

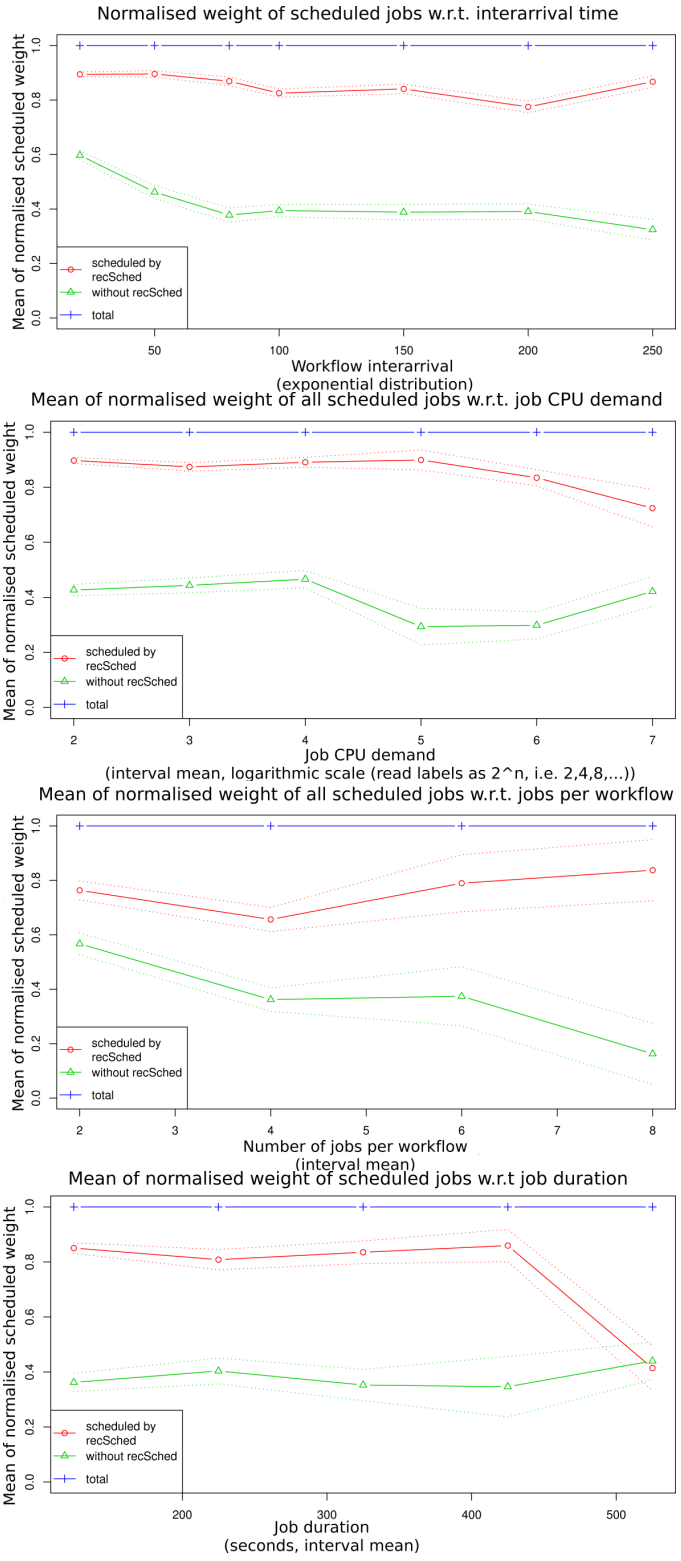


Figure 3. Simulation results from the prototype implementation show the pre-failure load, the load after applying our approach and the load without recovery. Dotted lines show the confidence interval to the 99% level around the mean form multiple measurements.

broken one: $D(S, S') = \sum_{t,r,j} \left| s_{t,r,j} - s'_{t,r,j} \right|$. The latter is a similarity measure for schedules and constitutes the connection between consecutive CSP solutions and eventually utilises the quality of the old schedule.

## IV. Evaluation

To evaluate our approach, the recovery scheme was implemented as a module of the grid management system VRM [1]. We simulated a Grid with 8 computational resources, providing between 32 and 512 CPUs each. Sequences of randomly generated workflows were scheduled to simulate the normal operation. Then, one of the resources was assumed to fail, to start the recovery mechanism.

We simulated a variety of parameters which we expected to impact the performance: load, job length, time available for recovery, number of jobs per workflow. The experiments in Figure 3 already show the significant advance of the recovery mechanism compared to naively dropping all affected future jobs and the workflows they belong to. In addition, the approach is often able to preserve more than 80% of the pre-downtime load.

All pictures show on the y-axis the preserved load normalized to the total load before the resource failure, with respect to varying parameters on the x-axis. The upper line is the total pre-downtime load itself – constantly one. The lower line is the load remaining in the schedule after the failure without any recovery mechanism. The middle line represents the proposed recovery scheme using DCSP.

The four experiments depicted in Figure 3 show that the performance of the recovery scheme is independent of the system parameter. By changing the interarrival time we modelled highly utilized grids (low interarrival time) and lower utilized Grids. In the upper right diagram we modified the complexity of the Grid workflows to be scheduled, which obviously has an impact at the loss if no recovery is done but hardly any impact on the recovery scheme. In the lower row we modified the size of the jobs in the resource and time dimension.

## V. Summary and outlook

We presented an approach to recover Grid workflows after resource failure. It focuses on the future jobs scheduled on the failed resource. Furthermore, we propose to employ a constraint solver for the search for optimal schedule. To this end, a formal representation of a schedule as an integer matrix is elaborated, as well as a constraint system which reflects the feasibility requirements for a schedule. The objective function for the optimisation includes maintaining similarity to previous solutions.

Future research will go in multiple directions: An alternative schedule representation can bring rigorous improvements in the complexity of constraint description and in the computational complexity. Study of the internal parameters of the approach, including the use of heuristics in the search phase, can lead to performance enhancements. Furthermore, formulation of the constraint in a portable way will allow for use of different constraint solvers and the strategies they implement.

## References


[1] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The virtual resource manager: An architecture for SLA-aware resource management," in *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid) 2004, Chicago, USA*, 2004, pp. 126 – 133.

[2] Burchard, L.-O. and B. Linnert, "Failure Recovery in Distributed Environments with Advance Reservation Management Systems," in *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), Davis, USA*, ser. Lecture Notes in Computer Science (LNCS), vol. 3278. Springer, 2004, pp. 112–123.

[3] D. Hollingsworth, "The workflow reference model version 1.1," Workflow Management Coalition, Tech. Rep. WFMC-TC-1003, January 19th 1995.

[4] MOAB Team, "MOAB workload manager," on-line, 2009, http://www.clusterresources.com/products/moab-cluster-suite.

[5] P. C. Inc., "Platform LSF," on-line, 2009, http://www.platform.com/products/LSFfamily.

[6] Altair Grid Technologies, "PBS Pro administrator guide 5.4," on-line, 2004.

[7] IBM, "Tivoli workload scheduler LoadLeveler," on-line, 2009, http://ibm.com/systems/clusters/software/loadleveler.

[8] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, p. 115, 1997.

[9] CCS Team, "CCS: Computing Center Software," on-line, 2009, https://www.openccs.eu/core/.

[10] R. K. Brett Bode, David M. Halstead and Z. Lei, "The portable batch scheduler and the maui scheduler on linux clusters," in *Proceedings of the 4th Annual Linux Showcase & Conference*. USENIX Association, 2000.

[11] S. Hwang and C. Kesselman, "Grid workflow: A flexible failure handling framework for the grid," *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03)*, 2003.

[12] L.-O. Burchard, B. Linnert, and J. Schneider, "A distributed load-based failure recovery mechanism for advance reservation environments." in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2, May 2005, pp. 1071–1078.

[13] R. Dechter and A. Dechter, "Belief maintenance in dynamic constraint networks," in *Proceedings of the 7th National Conference on Artificial Intelligence*, R. G. Smith and T. M. Mitchell, Eds. St. Paul, Minnesota: Morgan Kaufmann, Aug. 1988, pp. 37–42.

[14] Y. Ran, N. Roos, and J. van den Herik, "Methods for repair based scheduling," 2002, proceedings of the 21th workshop of the UK PLANNING and SCHEDULING Special Interest Group,PLANSIG 2002 (2002) 79-86.

[15] R. J. Wallace and E. C. Freuder, "Stable solutions for dynamic constraint satisfaction problems," in *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Maher and J.-F. Puget, Eds., vol. 1520. Springer, 1998, pp. 447–461.

[16] G. Verfaillie and T. Schiex, "Solution reuse in dynamic constraint satisfaction problems," in *Proceedings of the National Conference on Artificial Intelligence (AAAI 94)*, 1994, pp. 307–312.

[17] R. Bartak, "On-line guide to constraint programming, 1st edition; 1st part: Constraint satisfaction," on-line, 2009, http://ktiml.mff.cuni.cz/ bartak/constraints.

[18] Gecode Team, "Gecode: Generic constraint development environment," on-line, 2009, http://www.gecode.org.