Changes'05

International Workshop on Constraint Solving under Change and Uncertainty

A CP2005 workshop, Sitges, Spain
October 1st, 2005

Problem solving under change and uncertainty is a significant issue for many practical applications. Solutions must be obtained before the full problem is known, and often must be executed as the environment changes. Many of the application areas have been tackled by constraint based methods, but current constraint solving tools offer little support for uncertain dynamic problems. Possible enhancements could include rapid reaction to problem changes, robust solutions, prediction of future changes and contingent solutions, time guarantees or exploitation of known time limits.

This workshop will continue the series of CP workshops on online solving, change and uncertainty, and in particular following on from Changes'04 held at Toronto during CP2004. The workshop aims to bring together researchers interested in the general topic, to consider how existing techniques can be enhanced, and to explore combinations of different techniques.

This year, submission was by position paper only – this was intended to encourage participation and discussion at the workshop. All position papers are included in these working notes. We are very pleased to have an invited talk by Rina Dechter, on current research combining probabilistic reasoning with constraint networks. We welcome you to the workshop, and look forward to a fruitful discussion.

Workshop Organisers

Ken Brown	Cork Constraint Computation Centre, Ireland
Gérard Verfaillie	ONERA/DCSD, France

Advisory Committee

Chris Beck	University of Toronto, Canada
Neil Yorke-Smith	SRI International, USA

Acknowledgments

The organisation of this workshop was supported by grants SC/2003/81 (Enterprise Ireland), and 03/CE3/I405 (SFI Centre for Telecommunications Value-chain-driven Research).

Contents

Invited Talk: On Mixed Probabilistic and Deterministic Graphical Models

Rina Dechter

School of Information and Computer Science
University of California, Irvine, CA 92697
dechter@ics.uci.edu

Abstract. The talk will address algorithms for reasoning over knowledge-bases that can express both uncertain information and hard constraints. The need to accommodate both types of information is motivated by real-life applications, especially those involving planning and decision-making. Specifically, I will focus on the "mixed network", a new framework which integrates probabilistic networks and constraint networks.

Task Management under Change and Uncertainty

Constraint Solving Experience with the CALO Project

Pauline M. Berry, Karen Myers, Tomás E. Uribe, and Neil Yorke-Smith

Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA.
{berry,myers,uribe,nysmith}@ai.sri.com

Abstract The goal of CALO (Cognitive Assistant that Learns and Organizes) is to design an automated personal assistant to support a busy high-level knowledge worker. Operating in an inherently dynamic and uncertain domain, CALO relies on constraint technology in several components of its architecture. We outline the challenges and opportunities presented by constraint solving in the presence of change and uncertainty, embodied in CALO's personalized time management and task reasoning and execution systems.

1 Introduction

Consider the challenge of developing intelligent agents that assist a human by performing and managing tasks on her behalf. Such agents, operating in the real world, must handle change and uncertainty as a matter of course. This is the challenge of developing *CALO* (Cognitive Assistant that Learns and Organizes), an automated assistant hat will support a busy knowledge worker, such as a lab director or senior manager.[1] CALO should be able to perform routine office tasks for its user (e.g., arrange meetings, complete online forms, file email), manage open-ended processes (e.g., purchase a computer, arrange a conference), and anticipate and act on future needs of its user.

The cognitive assistant domain is inherently dynamic and uncertain. The user herself works in the evolving real world, with limited knowledge; with estimated, outdated or inaccurate information; interacting with colleagues; and encountering events outside her control, sometimes unpredicted. Moreover, users may change their minds, have inconsistent or mutually unachievable desires, and not be able or willing (or have the time) to communicate with their assistants.

In the following sections, we concentrate on two components of CALO where change and uncertainty impact the use of constraints technology: the personalized time management system, and the task reasoning and execution system.

Advantages of Constraints. We have found that using constraints in a system like CALO has a number of natural advantages. (1) Constraints have a well-defined, often intuitive semantics, which can be aligned with the ontology used in the system's knowledge base. (2) Constraints can capture qualitative and quantitative preferences and costs. (3) Constraints offer a declarative representation that is easy to produce and consume by different modules, including the human interface. (4) Constraints support notions of relaxation and explanation. (5) Finally, of course, constraints are supported by a large set of algorithms, solvers, and tools.

[1] www.ai.sri.com/project/CALO

2 Personalized Time Management

Time management is intensely personal. Many people, especially busy knowledge workers, are reluctant to relinquish control over the management of their own time. Moreover, individuals have different preferences and practices regarding how they schedule their time, how they negotiate appointments with others, and how much information they are willing to share when doing so. They also have different needs and priorities regarding the reminders they should receive.

The Personalized Time Manager (*PTIME*) assistant [1] is a CALO component with the goal of managing an individual's temporal commitments over an extended period of time, while recognising and adapting to the differences between individuals. Interaction between the human user and the system is central to this goal. The scheduling solutions found by the system should be informative, and the user–system dialogue should improve the quality of future interactions.

Requirements. At the heart of PTIME is a constraint-based scheduler. Solving the basic constraint problem is straightforward for small and medium instances, given present scheduling practice and available constraint technology. However, the basic problem is complicated by a number of factors, many due to change or uncertainty in one form or another. The following are some of the requirements for PTIME that go beyond pure constraint solving.

User requests At any moment the user can present new calendaring requests to PTIME. These include scheduling a new meeting, rescheduling existing ones, or adjusting locations and participants, while minimising perturbation on the user's schedule.

Preferences As described above, individuals have intense preferences over their time. These encompass not only when to schedule meetings—preferences on temporal and non-temporal constraints—but also the amount of autonomy the user permits PTIME. The user may not be able to articulate all her preferences, and may not be able or willing to communicate them to CALO; thus information is incomplete.

Execution Not everything occurs as planned: travel is delayed, meetings overrun, resources become unavailable. From the information CALO acquires about current execution, PTIME must reschedule, with the criteria of minimising perturbation and maximising likelihood of successful execution.

Relaxation Frequently, a meeting request cannot be satisfied without relaxing some constraints or violating some of the user's preferences.

Explanation An important requirement for CALO is that it be able to explain its own behaviour to its user. When a request cannot be satisfied, PTIME should be able to explain why this was the case, and present alternatives.

Interactivity The PTIME interface, shown in Figure 1, presents a view of the user's calendar, together with an interface for mixed-initiative decision making. For instance, if a meeting request is unsatisfiable, deciding whether and how to relax the request becomes a dialogue between the system and the user [2].

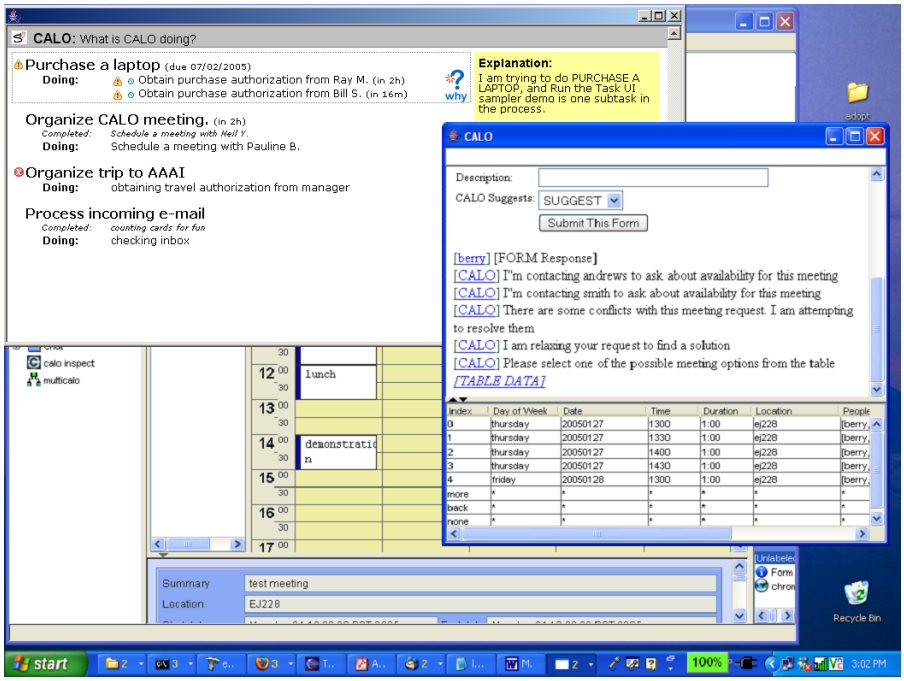


Figure 1. Development view of the PTIME user interface

Learning Over time, PTIME must learn and adapt to the user's preferences, which may themselves change. At present, PTIME can learn from explicit instruction ("I prefer morning meetings, except on Monday"), or implicitly, from the user's selection from a set of presented solutions to the CSP ("I prefer schedule A to schedule B") [3].

Distributed scheduling By its nature, satisfying a multi-person meeting request is a distributed negotiation problem over each potential participant. PTIME faces incomplete information on the preferences and schedules of others.

Solutions. The present constraint model in PTIME consists of prioritised finite domain constraints (FD), and disjunctive temporal constraints with semiring-based preferences (DTPP) [9]. There is a multi-criteria objective function, giving a soft constraint optimisation problem (COP). A sequence of COPs is solved, as the user interacts with the system to meet a calendaring request to her best satisfaction.

Although the COP is not large in size, and the mixed-initiative setting permits up to 500ms, say, for its solving, efficiency is important. This is due to the multiple COPs that must be solved for relaxation, explanation, negotiation, and rescheduling. Currently, solving is achieved by an FD-DTPP hybrid within a branch-and-bound search.

The present approach to change and uncertainty in PTIME is largely reactive. Our future work is to provide more robust schedules by proactive reasoning over the execu-

tion of tasks. In particular, we are considering learned probabilities on meeting occurrences and expected durations, and models of contingent events [6,11].

3 Task Reasoning and Execution

Ultimately, CALO assists its user by performing tasks on her behalf. These can range from keeping track of the user's commitments and providing reminders, to collaborating with the user in a mixed-initiative fashion, to fully autonomous completion of tasks explicitly delegated by the user, to proactive undertaking of activities that CALO determines will be beneficial (according to the permitted level of adjustable autonomy).

At the heart of CALO's ability to act is a *Task Manager* that initiates, tracks, and executes activities and commitments on behalf of its user, while remaining responsive to external events. The Task Manager component of CALO is based on a reactive execution system called *SPARK* [5].

SPARK is an agent framework grounded in a model of procedural reasoning. It is based on the Belief-Desire-Intention (BDI) model of agency [10], which has become the predominant architecture for the design of cognitive agents. SPARK has been developed to support the construction of large-scale, practical agent systems, and contains sophisticated mechanisms for encoding and controlling agent behaviour. At the same time, SPARK has a well-defined semantic model that is intended to support reasoning about the agents' knowledge and execution in a dynamic environment.

Constraint technology enables the temporal and resource reasoning of the Task Manager, together with parts of SPARK's deliberation over cognitive states. Both operate under the same requirements listed above for PTIME. Ongoing work includes:

Reasoning over commitments Action depends on knowing how to act, having the necessary means, and having sufficient time. Tasks are represented as hierarchical process models with simple deadlines. Richer temporal information will be represented as a Simple Temporal Network, augmented with information on expected (remaining) duration, probability of success, contingent events [6], and a profile of resource usage [4]. An important challenge is rapid incremental computation, suited to SPARK's reactive execution paradigm and the soft real-time demands on CALO's responses.

Guidance and goal selection Action follows decision on what to do. The Task Manager receives potential goals from the user, or raises them proactively. These, together with their deadlines and resource requirements, must be balanced against existing commitments in the light of user-stated *guidance* [8]. The various criteria may mutually conflict. The deliberation is modelled as a soft multi-criteria constraint optimisation problem over SPARK's cognitive states and the set of potential and current goals [7].

4 Conclusion

Change and uncertainty are real and pressing aspects for a cognitive assistant such as CALO. They arise not only from the situations in which CALO operates, but also from the demands of its user. Constraint-based models and constraint solving provide key

functionality for several components of the CALO architecture. In such a context, we must address dynamism, evolving requirements, incomplete knowledge, and ill-known outcomes head on. The case is made for sustained research in these areas, to develop more expressive CSP-based modelling frameworks and solving algorithms; and for systems that make these advances available in practice.

The cognitive assistant domain features other aspects beyond change and uncertainty that are just as demanding of the constraint solving. These include execution in the real world, mixed-initiative problem solving and interactivity, user preferences, learning, and distributed reasoning. Here again constraint technology is evolving. The case is made for an effort to provide constraint systems—whether languages or toolkits—that can help simultaneously address all these aspects.

Acknowledgement. We recognise all the members of the CALO project, and thank K. Conley for technical assistance. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

1. P. Berry, M. Gervasio, T. E. Uribe, M. E. Pollack, and M. D. Moffitt. A personalized time management assistant. In *AAAI 2005 Spring Symposium Series*, Stanford University, CA, Mar. 2005.
2. P. Berry, M. Gervasio, T. E. Uribe, and N. Yorke-Smith. Mixed-initiative issues for a personalized time management assistant. In *Proc. of ICAPS'05 Workshop on Mixed-Initiative Planning and Scheduling*, pages 12–17, Monterey, CA, June 2005.
3. M. T. Gervasio, M. D. Moffitt, M. E. Pollack, J. M. Taylor, and T. E. Uribe. Active preference learning for personalized calendar scheduling assistance. In *Proc. of the 2005 Intl. Conf. on Intelligent User Interfaces (IUI'05)*, San Diego, CA, Jan. 2005.
4. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003.
5. D. Morley and K. Myers. The SPARK agent framework. In *Proc. of the Third Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 714–721, New York, NY, July 2004.
6. P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proc. of IJCAI'01*, pages 494–502, Seattle, WA, Aug. 2001.
7. K. Myers and N. Yorke-Smith. A cognitive framework for delegation to an assistive user agent. In *AAAI 2005 Fall Symposium Series (to appear)*, Arlington, VA, Nov. 2005.
8. K. L. Myers and D. N. Morley. Policy-based agent directability. In H. Hexmoor, R. Falcone, and C. Castelfranchi, editors, *Agent Autonomy*, volume 7 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter 9. Springer, 2003.
9. B. Peintner and M. E. Pollack. Low-cost addition of preferences to DTPs and TCSPs. In *Proc. of AAAI-4*, pages 723–728, San Jose, CA, 2004.
10. A. S. Rao and M. P. Georgeff. Modeling agents within a BDI-architecture. In *Proc. of KR'91*, pages 473–484, 1991.
11. K. B. Venable and N. Yorke-Smith. Disjunctive temporal planning with uncertainty. In *Proc. of IJCAI'05*, Edinburgh, UK, Aug. 2005.

The Trading Agent Competition as a test problem for Constraint Solving under Change and Uncertainty

Kenneth N. Brown, David A. Burke, Brahim Hnich, Onur Koyuncu and Armagan Tarim

Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland
k.brown@cs.ucc.ie

1 Introduction

The annual Trading Agent Competition includes a Supply Chain Management game [1], in which agents must win contracts, obtain supplies, and produce goods, in competition with five other agents. In this position paper, we outline the problem domain, explaining why we believe it is a suitable testbed for reasoning about change and uncertainty in constraint optimisation, and we describe our initial attempts to design and implement a constraint-based agent to participate in the competition.

2 TAC SCM

The Trading Agent Competition Supply Chain Management games simulates a dynamic competitive supply and production environment. Multiple customers issue requests for quotes for the sale of PCs, including a quantity, a due date, and a lateness penalty. A number of production agents make bids for the contract, and for each request, a winning bid is selected. The winning agent must then manufacture the PCs, or provide them from stock, and ship them to the customer by the due date. In order to manufacture the PCs, the agent must procure raw material from suppliers. Again, requests for quotes (including delivery dates) are issued by the agent, the suppliers respond with offers, and an appropriate offer is selected. Each agent is limited by the capacity of its assembly line, and must select each day which set of products are to be manufactured. An agent has unlimited storage capacity, but must pay an inventory holding cost for each component and each finished PC. Failure to meet a due date on an order results in financial penalties. Each agent has an unlimited overdraft, but must pay interest at a rate higher than that received for a positive balance. The game consists of a year of 220 trading days, and each simulated day lasts for 15 seconds in real time. The aim of the game is to have the largest bank balance at the end of the year. The simulation is implemented in Java, and is controlled from a central server, to which remote agents must be connected. Basic Java agents are provided for

initial use. The competition consists of several rounds, and each round consists of many individual games. The participants in each round are ranked by their average closing bank balance.

The game clearly involves resource management, it is dynamic, and it involves significant uncertainty. There are in total 16 different PC configurations possible, each with a different set of components, and each requiring a different number of production cycles. The configurations are divided into three categories (*low, medium and high*) based on their expected price. Each day, the agent must decide which production cycles to devote to producing which configurations, based on the current set of actual and expected orders, and on the states of component and finished-goods inventory. Customer demand is uncertain - each day, the demand for configurations in each of the three cateogies varies based on a poisson distribution around a target which is varied as a random walk. The due dates and penalties (and maximum prices) are selected uniformly at random over an interval. Thus when scheduling production, an agent cannot know with certainty how many future orders will be available. Further, each day the agent makes offers, but does not receive results until the following day. The success rate will depend on the bids of the other agents. Thus each day the agent has an upper bound on the number of contracts which may be received tomorrow, but the actual contracts are not known with certainty until the following day. Daily reports on the maximum and minimum prices paid for each configuration are available. Every twenty days, the agent receives a market summary report, detailing the average prices. There is also uncertainty on the supplier side. An agent issues requests for the supply of components, and on the following day learns what offers have been made by the suppliers. The suppliers have limited resources, and will quote prices based on capacity, demand and the agent's reputation, and hence price depends both on random variables and on the behaviour of the other competing agents. Finally, the actual production capacity of the suppliers varies under a random walk from day to day, and so it is possible that the suppliers over-commit and deliver the components late.

Although the underlying constrained resource allocation problem is relatively simple, it is an online problem with uncertainty in the availability of resources. Further, the agents have some ability to make decisions on how the problem should change over time (by choosing which contracts to initiate), but that process is subject to significant uncertainty, caused partly by the game parameters, and partly by the presence of other agents acting in the game. Effective resource management therefore requires us to reason about both the changes and the uncertainty, and the competition provides a real and effective testbed.

3 Designing a constraint-based agent

We have designed and implemented an agent using Java and OPL, to compete in TAC-SCM 2005. Each trading day, we break the problem down into three decisions: (i) which PCs to manufacture, (ii) which components to request and order, and (iii) which PC orders to bid for and at what price. For (i), we are

initially taking a simple approach. We only schedule orders which have been confirmed, and each day, from the confirmed orders, we select jobs in order of due date that have the necessary components in stock, and schedule them until the production cycles are used up. For (ii), we attempt to order all components in advance. We maintain an expected order rate for each category of PC, which is updated each day based on the history of orders. Based on the quotes we received on the previous day, we make orders each day with long due dates to bring the component stocks up to the level required for the expected finished goods orders. In addition, we make short due date purchases to correct any errors in our previous estimates for both customer orders and supplier lead-times - that is, to bring stocks up to the levels required to fulfill confirmed orders.

Most of our effort has focused on deciding which bids to make each day, and on what prices to offer. We represent this as a constraint optimisation problem subject to uncertainty. Our aim is to select bids and prices to maximise our expected revenue, given component, inventory, lateness and interest costs, and given constraints on component availability and production capacity. However the success of any given bid depends on the actions of the other agents, and we are not given detailed information on their behaviour - we see only the minimum and maximum order prices for the previous day, and every twenty days we see the average order price for that period. Rather than try to reason explicitly about the behaviour of the other agents, we model the market conditions for each product as random variables, and we try to learn the probability of a bid price being accepted. We then use these probabilities in our optimisation model, and use expected production requirements and expected profits. Our aim is then to choose the bids and prices that maximise our expected profits, while ensuring that our expected demand does not exceed our capacity or supplies. It is possible that we succeed on too many bids, and then our production scheduling must decide which orders to satisfy.

This model is implemented in OPL Studio, and is embedded inside a Java agent. Each day, the agent updates its probability estimates for the success of different bids, its inventory, and its models of the suppliers; decides upon the day's production and delivery schedule; reads the RFQs from customers; runs the OPL model to make the bids; orders components from suppliers; and finally issues requests for quotes to suppliers for tomorrow's orders. The agent is currently participating in the competition, and appears to be robust and profitable, although not one of the leading agents. Future work will concentrate on improving the production scheduling and the management of supplies.

4 Conclusion

The Trading Agent Competition Supply-Chain Management game is a real time simulation of a competitive, dynamic and uncertain marketplace. As such, it provides an effective testbed for practical implementations of constraint solving under uncertainty. We have outlined our constraint-based agent for participat-

ing in the competition, and we will report on its final performance during the workhsop.

5 Acknowledgments

This work is supported by Science Foundation Ireland under Grant 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR).

References

1. J. Collins, R. Arunachalam, N. Sadeh, J. Eriksson, N. Finne and Sverker Janson "The Supply Chain Management Game for the 2005 Trading Agent Competition", Tech report CMU-ISRI-04-139, School of Computer Science, Carnegie Mellon University, 2004..

Open, Interactive and Dynamic CSP*

Santiago Macho González and Pedro Meseguer

Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
{smacho|pedro}@iiia.csic.es

Abstract. In previous work, the notions of Open, Interactive and Dynamic CSP have been independently defined. *Open* constraint satisfaction is a new model where values are incrementally gathered during problem solving. Domains are assumed unbounded. *Interactive* constraint satisfaction also deals with partially known domains, assuming implicitly that domains are finite. *Dynamic* constraint satisfaction deals with problems of dynamic nature (as configuration design or model composition) where variables, domains and constraints are subject to frequent changes. In this paper, we study the relationship between these three models, showing that Interactive CSP can be seen as a particular case of Open and Dynamic. We have applied two algorithms, FOCSP (developed for Open) and LC (developed for Dynamic) to solve Interactive CSP. We provide experimental results of this evaluation.

1 Introduction

In previous work, a new model called Open CSP [2], integrates information gathering and problem solving. This model starts solving a CSP from a state where all domains are possibly empty, and it dynamically asks for values while the CSP has not been solved. This process stops as soon as a solution is found. With a similar motivation, the Interactive CSP model [3] deals with problems with partially defined domains. It is implicitly assumed that the domains are finite, while in Open CSP domains remain unbounded. In a dynamic environment, tasks usually change. As a consequence, CSPs that represent these tasks evolve, and variables, domains and constraints may change over time. The Dynamic CSP model [1] was defined to solve CSP in such dynamic environments.

This paper considers the relation among Open, Interactive and Dynamic CSP approaches. We show that an Interactive CSP can be seen as a particular case of Open CSP, and also as a particular case of Dynamic CSP. Therefore, algorithms developed for Open or Dynamic models can be used for Interactive CSP.

The paper is organized as follows. First, we briefly describe Open, Interactive and Dynamic CSPs. Then, we show how Interactive CSPs can be seen as particular cases of Open and Dynamic CSPs, respectively. We provide an experimental evaluation of Interactive CSP using three algorithms, one from each class, on random problems, in terms of number of queries and search effort.

* Partially supported by the Spanish REPLI project TIC-2002-04470-C03-03.

2 Open, Interactive and Dynamic CSPs

Open CSP. Imagine you want to configure a PC using web data sources. Querying all the possible PC parts in all data sources on the web is just not feasible. We are interested in querying the minimum amount of information until finding a solution. Since the classical CSP approach (querying all values before search starts) is not applicable here, the new Open CSP approach [2] was proposed. Solving an Open CSP implies obtaining values for the variables, one by one. If the collected information does not permit to solve the problem, new values are requested. The process stops when a solution is found.

The *failure Open CSP (FOCSP)* algorithm [2] solves Open CSP, based on the idea that we gather new values only when the current known part of the problem has no solution. In that case, it contains a smaller subproblem that already has no solution, and the whole problem can be made solvable only by creating a solution to that subproblem. Then, an additional value should be found for one variable (the *failed* variable) of that subproblem, and search restarts. This algorithm is proven correct and complete, no matter unbounded domains.

Interactive CSP. Very related with Open CSP is the Interactive CSP model introduced by [3]. An Interactive CSP (ICSP) has partially known domains for its variables. When solving an Interactive CSP, new values are requested, until finding a solution or proving that no solution exists. The main difference with Open CSP is that Interactive CSP implicitly assumes that variable domains are finite.

In this model, it is possible to use heuristically some of the known constraints to guide the acquisition of new values. This feature depends on the concrete application to solve, but no specific condition is requested on the basic model. The *interactive forward checking (IFC)* algorithm is proposed. When a domain become empty, it launches a specific request for additional values that would satisfy the constraint on that variable. In this process, the algorithm may request all values of a variable, assuming finite domains.

Dynamic CSP. A Dynamic CSP [1] is a finite sequence $\langle \text{CSP}(0), \text{CSP}(1), \ldots \rangle$ of CSP instances, where each CSP(i) differs from the previous one by the addition or removal of some constraints (all possible changes of a CSP can be expressed in terms of constraint additions or removals). Solving a Dynamic CSP implies solving each instance of the sequence. The first instance is solved from scratch, and it is always possible to apply this method to any subsequent one. However, this approach is inefficient and may cause instability between solutions of consecutive instances. For this reason, when solving dynamic CSP one wants to reuse as much as possible the solving episodes of previous instances.

There are several solving approaches for Dynamic CSP solving. We consider the *local changes (LC)* algorithm [4], which tries to repair a previous solution. When the assignment of a variable becomes inconsistent with a previous solution, the inconsistent part of that solution is modified, keeping the assignment of that variable, until consistency is restored. If this is not possible, other values for the considered variable are tried.

3 Relationship between models

In this section we study the relationship between Open, Interactive and Dynamic CSP models. The relation between these 3 models is shown in figure 1.

Interactive CSP as Dynamic CSP. An Interactive CSP can be seen as a particular case of Dynamic CSP, as follows. In Interactive CSP, the operation that passes from a problem instance to the next one is *acquire_value*, getting a new value for a particular variable. Then, the variable domain is extended with that value, and the relational part of constraints involving such variable are enlarged with the allowed tuples that contain the new value. This process can be modelled in Dynamic CSP as follows. Adding a new value is equivalent to removing a unary constraint which disallowed this value in the domain of the corresponding variable, so that value is now available. Enlarging the constraints in which the variable is involved is equivalent to replacing (removing plus adding) the previous constraints by the enlarged ones.

At first glance one may think that this approach requires to know all the values of the domains from the beginning, to form the variable domains of the Dynamic CSP. However, this is not the case. It is enough to know the maximum number of values for each variable, say d_i for v_i. Initially, the problem state is as follows. The domain of v_i is a set of d_i dummy values $\{dummy_1, \ldots, dummy_{d_i}\}$. When value a is found, it replaces a dummy value, say $dummy_1$, in the variable domain (that now becomes $\{a, dummy_2, \ldots, dummy_{d_i}\}$), and in the constraints.

Strictly speaking, this model is an extension of the standard model of Dynamic CSP, where all domains are known from the beginning. The existence of dummy values which are replaced by real values as search progresses is not a big issue for the standard Dynamic CSP model, because the domain size does not change, and dummy values are replaced by real ones only once. This is the only extension that the standard Dynamic CSP model requires to include Interactive CSP.

Interactive CSP as Open CSP. An Interactive CSP can be seen as a particular case of Open CSP. Similarly to the Open CSP model, Interactive CSP uses partially defined data, where domains are acquired incrementally from external agents. The main difference between these models is that Interactive CSP does not address the problems of an open environment, in particular it limits itself to finite domains whilst Open CSP works with unbounded domains. Thus Interactive CSP instances can be solved with Open CSP algorithms.

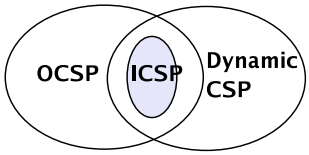
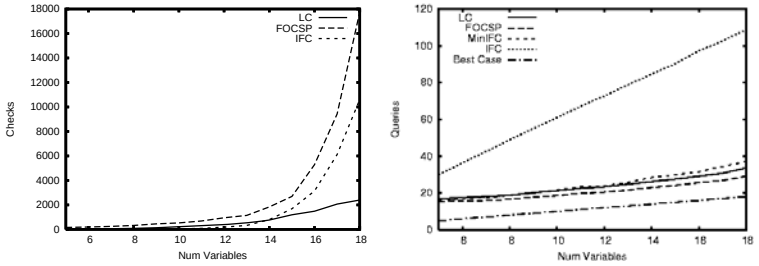


Fig. 1. *Relation between Open CSP, Interactive CSP and Dynamic CSP*

4 Experimental Results

We applied the LC algorithm [4] defined for solving Dynamic CSPs to solve Interactive CSP instances. Also we compare the FOCSP algorithm [2] defined for solving Open CSP problems which solves Interactive CSP problems. Both LC and FOCSP algorithms are compared with the IFC algorithm[3] developed for solving Interactive CSP problems.

To compare the algorithms, we are interested in the number of checks needed to solve the Interactive CSP and the number of accesses to information sources until a solution is found. We generated 1000000 random Interactive CSPs, with between 5 to 18 variables and 4 to 12 values per variable, with random constraints, forcing the graph to be at least connected and at most complete.



(a) (b)

Fig. 2. *(a) Comparison of the number of checks against the number of variables. (b) Comparison of the number of queries against the number of variables*

Figure 2(a) shows the number of checks against the number of variables, studying the performance of the algorithms when we increase the number of variables. It is shown that the LC algorithm has a much better performance that the FOCSP and the IFC algorithms. The FOCSP algorithm redoes again the same solving process every time a new value is added, the IFC is just a backtracking with forward checking, while the LC algorithm uses the information from the previous assignments (compatible assignments, incompatible assignments) for solving without solving again the CSP from scratch as the FOCSP algorithm does. The high number of checks of the IFC algorithm is due in part to the acquire_value phase, where the algorithm checks for a compatible value.

Figure 2(b) shows the number of queries needed to find a solution against the number of variables. We assume that a query retrieve just a single value. We can see that LC and FOCSP algorithms have a much better performance than the IFC algorithm which needs to collect all values in a mediator for doing the forward checking. Also we included the Minimal Interactive Forward Checking [3] algorithm (MinIFC) which queries compatible values without collect the complete variable domain. We notice that LC, MinIFC and FOCSP algorithms query nearly the same number of values to find a solution. We compare all algo-

rithms with the case where we found a solution just querying a value for every variable. We included this situation in the figure 2(b) with the line *Best Case*.

Figures 2(a) and 2(b) show the improvements of the LC algorithm applied to solve Interactive CSP problems. Empirically it is shown in figure 2(a) that reuse previous work on failed branches is better on average than detect the failed variable and redo the problem as the FOCSP algorithm does. Also the LC is better than backtracking with forward checking as the IFC algorithm does. It is interesting to analyze that the number of queries of LC algorithm is always slightly higher than the number of queries of FOCSP. This may be related with the way values are queried by both algorithms. While consecutive queries of FOCSP ask for values of different variables, consecutive queries of LC may ask the complete domain of a variable. Therefore, in some cases LC may ask more than needed to find a solution. This point is subject to current research.

5 Conclusions

In this paper we have analyzed the relation among Open, Interactive and Dynamic CSP. These models, different from the classical CSP, have appeared in different moments motivated by different applications. We have shown that Interactive CSP can be seen as a particular class of Open CSP (restricted to finite domains). In addition, we have also shown that Interactive CSP can be seen as a particular class of Dynamic CSP. As consequence, algorithms used to solve Open CSP and Dynamic CSP can be used to solve Interactive CSP. Based on this relationship, we have applied the FOCSP algorithm (initially developed for Open CSP) and the LC algorithm (initially developed for Dynamic CSP) comparing them with the IFC algorithm (developed for Interactive CSP) when they solve Interactive CSP instances. We have found that the LC algorithm reduces dramatically the number of checks with respect to FOCSP and IFC, just slightly increasing the number of queries needed to find a solution with respect to the FOCSP algorithm.

We think that this relationship between Open, Interactive and Dynamic CSP is a promising avenue for research, that we will further investigate.

References

1. Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh Annual Conference of the American Association of Artificial Intelligence*, pages 37–42, 1988.
2. Boi Faltings and Santiago Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161:181–208, 2005.
3. Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *IJCAI*, pages 468–477, 1999.
4. Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the Twelfth Conference of the American Association of Artificial Intelligence*, pages 307–312, 1994.

Optimal Solution Stability in Continuous-Time Optimization

Adrian Petcu and Boi Faltings

Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne (Switzerland)
{adrian.petcu, boi.faltings}@epfl.ch

Abstract. We define the *distributed, continuous-time combinatorial optimization problem*. We propose a general, semantically well-defined notion of *solution stability* in such systems, based on the *cost of change* from an already implemented solution to the new one. This approach allows maximum flexibility in specifying these costs through the use of *stability constraints*. We present the first mechanism for combinatorial optimization that guarantees optimal solution stability in dynamic environments, based on this notion of solution stability.
In contrast to current approaches which solve sequences of static CSPs, our mechanism has a lot more flexibility by allowing for a much *finer-grained vision of time*: each variable of interest can be assigned and reassigned *its own commitment deadlines*, allowing for a *continuous-time* optimization process. We emphasize that this algorithm deals with *dynamic problems*, where variables and constraints can be added/deleted at runtime.

Keywords:distributed AI, dynamic combinatorial optimization, solution stability

1 Introduction

Constraint satisfaction and optimization in a dynamic environment [2, 6, 7] is certainly an important aspect of constraint reasoning. As the world is generally dynamic, so are the problems that constraint algorithms must solve. Methods need to be devised to deal with such changes, and provide (good) solutions even when the problem has changed, and a previously determined solution is no longer useful. There is usually a certain amount of change that has to be inflicted upon an old solution for it to *adapt* to the new situation. Often it is impossible to change some parts of the old solution, or there is a high cost associated with change. Examples: resources are already consumed, re-routing trucks to serve a new customer costs extra fuel, re-assigning nurses to shifts in a hospital, etc.

Solution stability is an important concept that captures this ”change” of solutions in dynamic systems, and tries to minimize its effects.

2 Continuous time optimization

In dynamic systems, changes occur all the time, and optimization is a continuous process. In some cases, it is required to decide on the values of at least a subset of the

variables of the problem, and fix them to some optimal values. A simple example is a dynamic scheduling problem, where at some point one has to fix some tasks, and start working on them, otherwise the system would accomplish nothing but endless scheduling. Time has to be modeled explicitly, and taken into account.

Unfortunately, current approaches have a rather coarse grained vision of time: they assume the world holds still from one execution of the algorithm to the next, allowing them to solve static CSPs in each state, and minimize assignment changes from one solution to the next. This approach clearly lacks flexibility, and is not adequate in a multiagent system, because it means that all agents and tasks share the same rigid deadline.

In contrast, we propose a much finer-grained approach, where we assign *deadlines* to each variable of interest, thus allowing for a *continuous* solving process, with independent *commitment times* for each variable. Thus, depending on their semantics, some of the variables in the problem may be assigned such deadlines. Then, at the expiration of the deadline, these variables commit to their current optimal values. In case there is no need for a variable to be fixed at a given point in time, one can simply not assign any deadline to it, leaving it in a "floating" state, where it can always be freely assigned to its (current) optimal value.

We identify two kinds of commitments:

1. *Soft commitments* model contracts with penalties, and can be revised if the benefit extracted from the change outweighs its cost. Examples: a truck can be re-routed to a longer route if it picks up an additional package along the way, which yields an increase in revenue. A supplier may be willing to pay penalties for not delivering on time some goods to client X, if he gets a better revenue by serving customer Y first, etc. This kind of commitment can be modeled with *stability constraints* (see definition 1). We emphasize that deadlines for soft commitment can be re-specified at the expiration of the original ones. In fact, any variable in the system can have assigned to it a sequence of soft-commitment deadlines, possibly followed by a single hard-commitment deadline.
2. *Hard commitments* model irreversible processes, and are impossible to undo (example: production of good *X* already started, and resource *Y* was already consumed). When a hard commit is done, the variable is assigned to its committed value, it is marked "dead" and is logically eliminated from the problem. Physical elimination can be either immediate, or postponed for efficiency reasons, to be removed simultaneously with other *dead* variables. One can design a *garbage collection policy* to deal with dead variables. Hard commitments can be modelled as stability constraints with very large costs.

3 Solution stability based on cost

Current approaches like [6, 7, 1] define solution stability in dynamic CSP with respect to the number of variable assignments that need to be changed in order to reach again a consistent state upon a change in the problem. There are two approaches to achieve this kind of stability: first approach [6] is a curative approach: once a change occurs in the problem, they seek the new solution which is closest to the previous one, thus requiring

a minimal number of changes. The second approach [7, 1] is a proactive approach: when generating a solution in the first place, one tries to find robust solutions, which are likely to remain valid even upon changes in the problem, thus requiring little or no adjustment. [1] uses a probabilistic model that tries to predict what possible changes can happen in the future, and tries to generate solutions that are robust with respect to the predicted changes.

We break away from this definition of stability by looking at the process from a *cost* perspective. We argue that actually the number of assignments that change is irrelevant; what matters is the total *cost* that is induced by these changes, once the assignments are made. If these costs are outweighed by increases in solution quality, one can gain from making as many changes as necessary. In the truck routing example, if rerouting a few trucks means serving a new task that produces a benefit which far exceeds the additional fuel costs, then we reroute the trucks, and gain the benefits. Consider also a scheduling problem where a company makes a schedule for a lot of small interconnected tasks that lead to the assembly of a product which is supposed to be delivered at a specified time. The manufacturing process starts, but there is a change (e.g. a machine breaks down). Keeping the old schedule valid as much as possible (as it happens when stability is defined as minimizing assignment changes) may lead to the final delivery date being pushed back for 2 months, thus costing the company big penalties. Working from the cost perspective can mean that almost all tasks from the old schedule could be rescheduled, but the final objective (on time delivery) is attained.

We introduce these costs of changing variable assignments through *stability constraints* in Definition 1. Notice that with this definition, one can accurately specify not only that it costs something to change one assignment, but also how much it costs to replace it with any other assignment. In practice, this allows for accurate modeling of all possible contingencies. For example, it is obvious that re-routing a truck through route A is more expensive if the truck is now located in a city far away, than if the truck were located in a nearby city.

Definition 1. *A **stability constraint** σ_i is a pseudo-binary constraint on X_i. The semantics of such a constraint is simple: if X_i is assigned to v_i^1, then $\sigma_i(v_i^1 \rightarrow v_i^2)$ denotes how much it costs to change X_i's value to v_i^2.*

Definition 2. *Formally, a discrete **continuous time multiagent constraint optimization problem** (CMCOP) is a tuple $< \mathcal{X}, \mathcal{D}, \mathcal{R}, \mathcal{S}, \mathcal{T} >$:*

- *$\mathcal{X} = \{X_1, ..., X_m\}$ is the set of variables/solving agents;*
- *$\mathcal{D} = \{d_1, ..., d_m\}$ is a set of domains of the variables, each given as a finite set of possible values.*
- *$\mathcal{R} = \{r_1, ..., r_p\}$ is a set of relations, where a relation r_i is a function $d_{i1} \times .. \times d_{ik} \rightarrow \mathbb{R}^+$ which denotes how much utility is assigned to each possible combination of values of the involved variables.*
- *$\mathcal{S} = \{\sigma_1, ..., \sigma_m\}$ is a set of stability constraints*
- *$\mathcal{T} = \{t_1, ..., t_m\}$ is a set of commitment deadlines: times until the corresponding variable has to commit to a value. Deadlines can be for hard or soft commitments.*

Formally, we define the new optimal solution like this:

$$\mathcal{X}^*_{new} = argmax_{\mathcal{X}} \left(\sum_{r_l \in \mathcal{R}} r_l(\mathcal{X}) - \sum_{\sigma_i \in \mathcal{S}} \sigma_i(\mathcal{X}^{old} \rightarrow \mathcal{X}) \right) \quad (1)$$

where the first sum is the utility of the new solution, and the second sum is the cost one has to pay for changing the current assignments to the new ones.

For uncommitted variables, the cost is 0: they can simply choose their new optimal values, without any cost. Hard-committed variables cannot change their values anymore (one can think of it as an infinite change cost).

Thus, what we need to optimize is the difference between the new utility and the cost associated with changing the soft-committed variables.

4 RSDPOP: a self-stabilizing protocol for MCOP

For dynamic environments, self stabilizing algorithms([3]) are particularly well suited, since they guarantee some desired behavior even when there are changes in the problem. The *RSDPOP* algorithm from [4] is a self-stabilizing algorithm that works for dynamic, distributed *CMCOP* problems. *RSDPOP* guarantees stabilization in the optimal solution of the optimization problem. It is composed of 3 concurrent self-stabilizing protocols, that are initialized and then run concurrently:

- self-stabilizing protocol for DFS tree generation: its goal is to create and maintain (even upon faults/topology changes) a DFS tree maintained in a distributed fashion
- self-stabilizing protocol for propagation of utility messages: bottom-up utility propagation along the DFS tree
- self-stabilizing protocol for propagation of value assignments: based on the utility information obtained during the previous protocol, each node picks its optimal value and informs its children (top-down along the DFS tree).

RSDPOP is actually an extension of a basic self-stabilizing algorithm for distributed optimization (*SDPOP*, see [5]). *RSDPOP* basically adds solution stability reasoning to *SDPOP*.

For detailed descriptions, correctness proofs, complexity analisys and experimental results, please refer to [5, 4].

5 Experimental evaluation

We performed experiments on distributed meeting scheduling problems. We modeled a realistic scenario, where a set of agents try to jointly find the best schedule for a set of meetings. We assume that the agents must travel to the meetings they will attend. Thus, they must arrange for transportation (airplane tickets) and accommodation (hotel reservations). There is obviously a cost for canceling these arrangements, as some tickets may be non-refundable, some hotels may charge the room price for one night in case

of no-show, etc. This is modeled by a stability constraint on each variable whose reassignment involves a cost. We simulate a dynamic problem by changing the preferences of the agents dynamically.

The task is to find the best possible schedule at each time, taking into account the new preferences of the agents, but also the cancellation costs that are incurred by changing the starting times of the meetings that the agents have committed to. The results of these experiments can be found in [4].

6 Conclusions and future work

We define the *distributed, continuous-time combinatorial optimization problem*. We propose a general, cost-based metric for *solution stability* in such systems. We present the first mechanism for combinatorial optimization that guarantees optimal solutions in dynamic environments, with respect to this metric. In contrast to current approaches, our mechanism is a lot more flexible and allows for a much finer-grained vision of time: each variable of interest can be assigned *its own commitment deadlines*, allowing for a *continuous-time* optimization process. The experimental results show that this approach gives good results for low width, practically sized dynamical optimization problems.

As future work we envisage addressing issues like robustness, different garbage collection policies, and analyzing ways of dealing with very tight deadlines.

References

1. Ken Brown and David Fowler. Extending constraint programming with uncertain reasoning for robust dynamic scheduling. In Ken Brown and Chris Beck, editors, *CUW'01: Constraints and Uncertainty workshop at CP'2001,* Cyprus, November 2001.
2. A. Dechter and R. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-88*, pages 37–42, St. Paul, MN, 1988.
3. Shlomi Dolev. *Self-Stabilization.* MIT Press, 2000.
4. Adrian Petcu and Boi Faltings. Optimal solution stability in continuous time optimization. In *IJCAI05 - Distributed Constraint Reasoning workshop, DCR05*, August 2005.
5. Adrian Petcu and Boi Faltings. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, Pittsburgh, Pennsylvania, USA, July 2005.
6. G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-94*, pages 307–312, Seattle, WA, 1994.
7. R.J. Wallace and Eugene C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proceedings of CP98,* 1998.

Looking for a common scheduling perturbations benchmark

Nicola Policella and Riccardo Rasconi *

ISTC-cnr
Institute for Cognitive Science and Technology
National Research Council of Italy
{name.surname}@istc.cnr.it

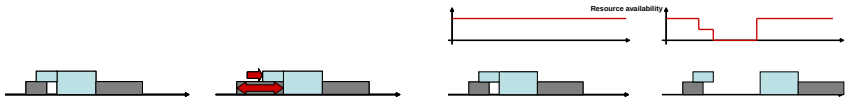
In the last workshop on Constraint Solving under Change and Uncertainty, emerged the general lack of common benchmarks on which to test algorithms and strategies to solve the Constraint Satisfaction Problem in dynamic and/or uncertain environments. This paper represents an attempt to meet this urgency. In particular, we analyze the creation of benchmark data sets for the scheduling problem which, as well known, represents a significant CSP instance.

Scheduling is defined in theory as the problem of assigning start and end times to a set of activities (or tasks) subject to a number of constraints. Yet, the synthesis of initially feasible schedules is hardly ever sufficient, as in real-world working environments, unforeseen events tend to quickly invalidate the schedule predictive assumptions and bring into question the continuing consistency of the schedule's prescribed actions. In the last years a number of works aimed at facing this uncertainty by proposing different scheduling techniques: from reactive approaches [1, 2] to robust solutions [3, 4] and partially defined solutions [5, 6]. For a fair comparison of all the different methodologies, it is necessary to use a common empirical framework, composed of benchmark sets and proper evaluation metrics. In particular, in this work we focus on the production of a benchmark generator for instances on scheduling perturbations. This is a necessary instrument to assess the validity of the various rescheduling methodologies. In fact, the presence of this benchmark might reveal crucial to boost research on scheduling.

1 Scheduling perturbations

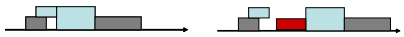
In the case of the scheduling problem, the uncertainty aspects which normally permeate the physical environments have to be properly modeled through the characterization of a set of particularly significant exogenous events; the benchmarks we are pursuing to develop will be based on the production of sequences of elements taken from this set. Real world uncertainty can be singled out in the following points: *activity delay*, e.g., a surgery operation must be delayed until the doctors arrive; *growth of activity processing time*, e.g., getting a flat tyre inevitably extends the duration of the journey; *lowering of resource availability*, e.g., an unexpected loss of a piece of machinery in an assembly line; *variation in the number of activities*, e.g., adding an unscheduled visit to the mother in law in the daily plan; *change in the mutual ordering of the activities*, e.g., an activity in a production chain may suddenly become more urgent than another.

* PHD Student at the Department of Informatics, Systems and Telematics - University of Genova

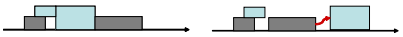


(a) Temporal changes: activities can last more than expected or they can be postponed until necessary conditions are satisfied.

(b) Resource changes: the two red curves represent the nominal (left) and the actual (right) resource availability. The reduction of resource availability blocks the execution of the last two activities, which are delayed.



(c) Activity changes: the need to serve a new activity requires a reallocation of the current scheduled activities.



(d) Causal changes: a new precedence relation between a pair of activities requires a revision of previous choices.

Fig. 1. Different events may require interventions to re-establish the validity of the schedule

Therefore, the elements of uncertainty which may normally affect the consistency of a schedule basically belong to one of the following types: (1) temporal changes, which involve the various temporal aspects of the problem; (2) resource variations, which modify the resource availability during the execution of a schedule; (3) causal changes, which involve the introduction of new constraints among the activities and/or new activities. Figure 1 shows how the events described above can affect the schedule during its execution.

1.1 Building blocks of a benchmark generator

In the production of a benchmark set for the previous problem, a key point is to recognize the types of unexpected events which can spoil the execution of the solution, their magnitude, as well as and their relative temporal spacing. In this section we discuss how these aspects have been taken into account during the production of our benchmark generator.

The importance of event spacing. Event spacing is determined through the production of events each characterized by a particular value of the parameter t_{aware}: this parameter specifies the "absolute" instant where each specific event occurrence is acknowledged (and reacted upon). The temporal distance between the actual time of execution and t_{aware} can be exploited by the scheduler to accommodate the activities for rescheduling. Once all the events are generated, the t_{aware} parameter will be used

to temporally sort all the generated events so that they can be "fired" according to the correct order of occurrence.

Great attention must be paid in order to produce t_{aware} values that are guaranteed to be valid for any possible schedule execution: infact, it is not unlikely that a produced event, valid for a given solution, loses its validity as a consequence of a rescheduling decision which alters the previous solution. In [7] we consider this issue and identify a number of problem relaxations and assumptions which guarantee t_{aware} independence from any scheduling decision. It is possible to argue that a viable alternative could be to synthesize "relative" values for the t_{aware} parameter instead of absolute values, for instance by relating them to the start times of the activities they refer to. Unfortunately this choice is not applicable, as it can lead to unfeasible situations where some event, supposed to be fired on one activity according to t_{aware}, may not face any possible introduction, i.e. its value may fall into the past w.r.t. the current execution time (see [7]). Moreover, introducing relative t_{aware} values might yield an extremely undesirable effect: each single instance produced by the benchmark generator might result in different executional behaviors w.r.t. the considered schedule[1], depending on the particular decisions taken by the rescheduling engine.

Definition of the different exogenous events. In order to define a benchmark set for the dynamic sub-problem, we refine here the concept of "event" introduced above. For each exogenous events we provide in the following a detailed definition.

– *delay of an activity*, e_{delay}:

$$e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$$

besides the activity to be delayed a_i and the width of the shift, Δ_{st}, it is necessary to specify the instant where the specific event is acknowledged by the execution monitoring system, t_{aware} (this is a common element of all the defined events);

– *change of an activity duration*, e_{dur};

$$e_{dur} = \langle a_i, \Delta_{dur}, t_{aware} \rangle$$

like the previous case it is necessary to specify three different parameters: the activity a_i, the change in duration Δ_{dur}, and t_{aware};

– *change of a resource availability*, e_{res};

$$e_{res} = \langle r_j, \Delta_{cap}, st_{ev}, et_{ev}, t_{aware} \rangle$$

in this case, there are more parameters to specify: the resource involved r_j, the variation in resource availability Δ_{cap}, the time interval in which the change takes place $[st_{ev}, et_{ev}]$, and t_{aware}. We note that the time interval can be infinite, i.e. $et_{ev} \rightarrow \infty$;

– *change of the set of activities to be served*, e_{act}

$$e_{act} = \langle f_a, a_k, \overline{req_k}, dur_k, est_k, let_k, t_{aware} \rangle$$

where the parameter $f_a \in \{add, remove\}$ is a flag that describes whether the activity a_k has to be added or removed; $\overline{req_k} = \{req_{k1}, \ldots req_{km}\}$ is an array that specifies the

[1] In fact, a scheduling problem admits in general several solutions.

required capacity for each involved resource. Then the activity duration dur_k, the time interval in which this activity has to be served $[est_{a_k}, let_{a_k}]$ and t_{aware}. Of course we have that $let_k - est_k \geq dur_k$.

– insertion or removal of a causal constraint between two activities, e_{constr}

$$e_{constr} = \langle f_c, a_{prec}, a_{succ}, d_{min}, d_{max}, t_{aware} \rangle$$

where the flag $f_c \in \{add, remove\}$ describes if the constraint between a_{prec} and a_{succ} has to be posted or removed. We need also to specify the minimum and maximum distance d_{min}, d_{max} imposed by the constraint and t_{aware}. In case the Δ_{st} parameter of the e_{delay} event should be negative, this reflects in starting the activity earlier than expected. Similarly, a negative value for Δ_{dur} in the e_{dur} event determines an early stop of the activity, while a negative value of Δ_{cap} determines an increase of the resource availability during the interval $[st_{ev}, et_{ev}]$.

Regarding the last two events, we have to say that in case of activity and/or constraint removal (e_{act} and e_{constr}) it is necessary only to specify the parameters related to the involved activities, that is a_k and the pair (a_{prec}, a_{succ}), respectively.

1.2 Scheduling perturbations as a dynamic CSP

As the scheduling problem represents a significant instance of the Constraint Satisfaction Problem (CSP), the uncertainty aspects described above represent challenging features for a dynamic CSP. In the following we recall the CSP paradigm, how this can be used to model a scheduling problem, and how the presence of unforeseen events transforms the problem into a dynamic CSP.

According to CSP paradigm, a scheduling problem can be formulated in the following way: a set of variables, or time points, t_i is introduced; they represent the temporal events of the problem, that is, the start time and/or the end time of each activity. For each variable t_i a domain $D_i = [0; H]$ is assigned (where H is an upper bound on the scheduling horizon), which specifies the validity range for t_i's values. Two types of constraints combine to further restrict the values that may be assigned to the set of variables: (1) binary constraints (involving pairs of variables) for representing the temporal relations between activities (i.e., activity durations, precedence between pair of activities); and (2) n-ary constraints to describe the capacity constraints that each resource imposes on all feasible schedules.

A dynamic constraint satisfaction problem (DCSP) is a generalization of the constraint satisfaction problem. In [8] it has been defined as a sequence of CSPs where each differs from its predecessor by constraints addition or deletion. In our case though, the CSPs may differs also by the number of involved variables. In particular, the elements of uncertainty listed above naturally create a sequence of CSPs. For instance, an activity delay is modeled through posting a new constraint between the source of the scheduling problem and the activity of interest. If activity a_i has to be delayed from st_i to $st_i + \delta_i$, a constraint labeled $[st_i + \delta_i, H]$ is posted. Another example is how to reduce resource availability: this is modeled indirectly, through the insertion of a "ghost" activity which requires a certain amount of resource capacity: the required amount will be equal to the the capacity reduction we want to model, the duration of the activity will be equal to the resource unavailability interval (which can be infinite).

2 Conclusions and future work

Our research work aims at analyzing and producing benchmark data sets for the scheduling execution problem. This effort is justified by the absence of such benchmarks and by our conviction that an experimental analysis of the execution of schedules can reveal invaluable to assess the effectiveness of different approaches to scheduling problems. To this aim, we propose a benchmark based on the production and firing of a variable number of events chosen from a predetermined set, aimed at testing the effectiveness of rescheduling algorithm as well as the robustness of the initial schedule.

In order to design a complete experimental framework, it is essential to introduce also a set of metrics to evaluate the validity of the different rescheduling techniques by producing an assessment of the quality of the solution according to various criteria. The "dynamic" optimization criteria are in general different then those related to the static case, as schedule execution imposes the presence of different requirements. Moreover, given the generally strict time availability over which the schedule revision procedure is called to react, sometimes solution quality must come as a secondary priority as the execution of the schedule does not allow for time-intensive computations.

An important measure is represented by the schedule *continuity* (or *s*tability), which may informally be described as the closeness of the perturbed schedule to the schedule before the occurrence of the disturb. Solution continuity can be a very important quality measure of the schedule: in many cases it is in fact essential that any revised solution be as close as possible to the previous consistent solution found by the scheduler; the closer any two solutions are to each other, the higher their level of continuity.

Regarding the benchmark generator, we are currently producing and running the first experiments. We plan to terminate the tuning phase soon and to make available the generator on the web in a few months [2].

References

1. Smith, S.F.: OPIS: A Methodology and Architecture for Reactive Scheduling. In Fox, M., Zweben, M., eds.: Intelligent Scheduling. Morgan Kaufmann (1994)
2. El Sakkout, H.H., Wallace, M.G.: Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. Constraints **5** (2000) 359–388
3. Leon, V., Wu, S.D., Storer, R.H.: Robustness measures and robust scheduling for job shops. IIE Transactions **26** (1994) 32–43
4. Beck, J.C., Wilson, N.: Proactive Algorithms for Scheduling with Probabilistic Durations. In: Proceedings of the 19^{th} Int. Joint Conference on Artificial Intelligence, IJCAI-05. (2005)
5. Wu, S.D., Beyon, E.S., Storer, R.H.: A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. Operations Research **47** (1999) 113–124
6. Policella, N.: Scheduling with Uncertainty: A proactive approach using Partial Order Schedules. PhD thesis, Università di Roma "La Sapienza" (2005)
7. Policella, N., Rasconi, R.: Testsets generation for reactive scheduling. In: Proceedings of Workshop on Experimental Analysis and Benchmarks for AI Algorithms. (2005)
8. Dechter, R., Dechter, A.: Belief Maintenance in Dynamic Constraint Networks. In: Proceedings of the 7^{th} National Conference on Artificial Intelligence (AAAI'88). (1988)

[2] `http://pst.istc.cnr.it/`

Change and Uncertainty in Temporal CSPs

Kristen Brent Venable[1] and Neil Yorke-Smith[2]

[1] University of Padova, Italy. kvenable@math.unipd.it
[2] Artificial Intelligence Center, SRI International, USA. nysmith@ai.sri.com

Abstract Temporal reasoning, by its nature, is bound up with change and uncertainty. Underestimating such factors can lead to unrealistic solutions, in practice not executable due to their lack of robustness to unpredictable events. Constraint Satisfaction Problems (CSPs) have shown their effectiveness both in modelling and solving temporal problems. Classical temporal CSP frameworks such as the Simple Temporal Problem have been extended with the notion of uncontrollable variables, to provide more realistic solutions by accommodating contingent events outside the control of the executing agent. Even so, for many applications this extension is not sufficient by itself, such as when the problem also features preferences, soft constraints, stochastic uncertainty, or disjunctive choices, among other factors. We outline several recent modelling advances to combine some or all of these aspects in a single temporal CSP formalism.

1 Simple Temporal Problems with Uncertainty

Constraint Satisfaction Problems (CSPs) have shown their effectiveness both in modelling and solving temporal problems. An outstanding example that has been successfully and widely applied in practice is the quantitative formalism known as the *Simple Temporal Problem* (STP) [3]. In an STP, variables X_i represent time-points, and constraints describe binary temporal differences of the form $a_{ij} \leq X_j - X_i \leq b_{ij}$, i.e. $X_j - X_i \in [a_{ij}, b_{ij}]$.

Temporal reasoning, by its nature, is bound up with change and uncertainty. To address one aspect of these factors, namely contingent events outside the control of the executing agent, the STP has been extended with the concept of uncontrollable variables. In the resulting framework, the *STP with Uncertainty* (STPU) [23,10], the notion of *controllability* replaces the usual notion of consistency. Solving a problem with uncertainty is redefined in terms of finding solutions with certain degrees of robustness to uncertainty. The most important of these controllability levels, dynamic controllability, is tractable in polynomial time [10,9].

The STPU has been applied in practice in scheduling and planning problems, in systems such as IxTeT [4]. Nonetheless, by itself the model accommodates only one aspect of uncertainty. In the following sections we outline extensions that address other aspects of change and uncertainty, and extensions that address still other factors, such as preferences, that arise in many applications. We argue that constraint programming must develop formalisms and methods that simultaneously address all these factors.

2 Change and Uncertainty Beyond the STPU

The solution produced for an STPU depends on the level of controllability that can be established for the temporal network [23]. We might crudely characterise the situation as follows: a strongly controllable STPU gives a fully robust solution, a dynamically controllable STPU gives a flexible solution, and a weakly controllable STPU gives a contingent solution. In each case the solution is prepared offline.

On executing a schedule or plan described by an STPU solution, the executing agent may encounter contingencies not handled in the solution produced by the offline STPU solving process, whichever notion of controllability is used. As examples, consider dynamic task sequencing in a STPU for tasks with resources [22], and incremental updating of an STP upon unanticipated contingent events [5].

Even for events that can be anticipated, the contingent policy that is guaranteed to exist by the dynamic controllability property of an STPU has been found to be insufficiently expressive. For example, whether an autonomous spacecraft can undertake a manoeuvre may depend on whether a minimal level of energy is available. Therefore several frameworks add conditional branching based on the occurrence of events [8,20].

The STPU assumes that uncontrollable variables take values within their domains, which are fixed, a priori known intervals. Controllability properties either hold or not. This is an unrealistic model for some applications. For example, how long does an image downlink from the spacecraft take? If we set the interval for this contingent duration narrowly, the STPU is more likely to be controllable but the risk is greater that the supposed bounds on the downlink will be violated. On the other hand, if we set the interval broadly, the STPU is likely to be uncontrollable, providing us with no information. Either way, the problem modeller cannot readily express the 'likely' duration.

To address this issue more realistically, the *Probabilistic STP* [18] attaches a probability density function to each uncontrollable variable. Subsequently, controllability properties hold with a certain probability. Determining which levels of controllability hold with what probability for a PSTP, however, is computationally much harder than determining controllability for an STPU [19].

Moreover, the STPU partitions the variables into controllable and uncontrollable with the semantics that assignment of values to each time-point is controlled respectively by the executing agent or by exogenous factors summed up as 'Nature'. In a distributed agent setting, there may in fact be multiple agents who are jointly executing a plan. In this case, the model of controllability must be extended beyond the two actors of the STPU [6].

3 Enhanced Expressiveness in the STPU

While the STP is attractive because of its tractable computation, its expressiveness is limited. As an example, consider a scenario where Alice is scheduling a one-hour meeting with a visitor, Bob, and would like at least two engineers to attend. On the afternoon that Bob is visiting, Alice would like to exercise sometime between noon and the end of the day at 6:00 p.m, has a seminar scheduled from 2:00 to 3:30 p.m, but is otherwise free; Bob must leave for the airport by 4:00 p.m. Engineer Carl is free after 1:00 p.m,

and engineer David is available all afternoon. Alice's gym session is at least 40 minutes, and if it occurs before a meeting or seminar, she needs another half an hour to cool down beforehand.

This type of time management scenario features contingent events, such as the actual duration of the seminar. In fact, Alice might well hold some probability distribution of how long she thinks the seminar will last, based on her prior experience of the speaker. Beyond contingent events, the scenario also features the action of multiple agents with individual schedules (Carl and David have autonomous interests and actions from Alice), conditional constraints (e.g. if the gym session is before the seminar, the schedule must include cool-down time), disjunctive constraints (e.g. the gym session can be before or after the meeting), preferences (e.g. a minimum time at the gym, but a longer session is better, up to an hour). One can imagine readily additional complications in real life.

To accommodate disjunctive constraints, such as "event A occurs before or after event B", the *Disjunctive Temporal Problem* (DTP) [17] permits disjunctions of STP constraints. To accommodate user preferences, the *Simple Temporal Problem with Preferences* (STPP) attaches a semiring-based preference function to each constraint [7]. In both approaches, effective solving methods have been developed, at least in important cases. The two extensions have been combined in the *DTPP* [13].

The challenge that arises is to develop frameworks that account simultaneously for these factors and for change and uncertainty, and to develop effective solving methods for them. One step in this direction is the *DTP with Uncertainty* (DTPU) [21]. The DTPU combines disjunctions of STP constraints with controllable and uncontrollable variables. This developing work raises both semantic and algorithmic issues that we are now exploring.

To both express user preferences and handle contingency, two related temporal CSPs have been proposed. The *Simple Temporal Problem with Preferences and Uncertainty* (STPPU) [15] combines the STPP and STPU, while the *Simple Temporal Problem with Preferences and Probabilities* (STP^3) [11] combines the STPP with the PSTP. In both cases, non-stochastic and stochastic, we must find a meaningful, principled way to combine the optimisation component arising from the preferences with the controllability notions that address the contingency (compare [14]); and then, once again, develop efficient solving methods. Our approach has been to leverage solvers for the component frameworks, as far as possible.

4 Looking to the Future

The effectiveness of constraint programming for temporal reasoning is now established in mature and widely-applied frameworks such as the Simple Temporal Problem. As the scheduling literature shows [2], the relevance of the solutions obtained depends on their executability in the real world of change and uncertainty. Work must continue to extend the STP to accommodate contingency, probabilities, evolving problem formulation, online execution, and related aspects, if we are to generate practical solutions.

Second, temporal problems in the real world contain important factors orthogonal to uncertainty. Work must continue to extend the STP to include such factors as pref-

erences, soft constraints, and disjunctive constraints. The challenge then is to develop frameworks that account simultaneously for these factors and for change and uncertainty, and to develop effective solving methods for them.

Our specific work is in developing models and methods for such a combination of factors. Motivational problems from domains such as aerospace planning and assistive agents [1,16,12] mean that temporal CSPs must ultimately address all the aspects discussed.

Acknowledgments. We thank M. Moffitt, R. Morris, M. Pollack, F. Rossi, and T. Uribe. The work of the last author is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

1. J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. of UAI-02*, pages 77–84, Aug. 2002.
2. A. J. Davenport. Managing uncertainty in scheduling: A survey. Technical report, Enterprise Integration Laboratory, Dept. of Mechanical and Industrial Engineering, University of Toronto, 1998.
3. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991.
4. M. Gallien, F. Ingrand, and S. Lemai. Robot actions planning and execution control for autonomous exploration rovers. In *Proc. of ICAPS'05 Workshop on Plan Execution*, pages 33–41, Monterey, CA, June 2005.
5. I. hsiang Shu, R. Effinger, and B. Williams. Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *Proc. of ICAPS'05*, pages 252–261, Monterey, CA, June 2005.
6. L. Hunsberger. Distributing the control of a temporal network among multiple agents. In *Proc. of AAMAS'03*, pages 899–906, Melbourne, Australia, 2003.
7. L. Khatib, P. Morris, R. A. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. of IJCAI'01*, pages 322–327, Seattle, WA, 2001.
8. P. Kim, B. C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. of IJCAI'01*, pages 487–493, Seattle, WA, 2001.
9. P. Morris and N. Muscettola. Temporal dynamic controllability revisited. In *Proc. of AAAI-05*, pages 1193–1198, Pittsburgh, PA, July 2005.
10. P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proc. of IJCAI'01*, pages 494–502, Seattle, WA, Aug. 2001.
11. R. Morris, P. Morris, L. Khatib, and N. Yorke-Smith. Temporal planning with preferences and probabilities. In *Proc. of ICAPS'05 Workshop on Constraint Programming for Planning and Scheduling*, pages 52–57, Monterey, CA, June 2005.
12. K. Myers and N. Yorke-Smith. A cognitive framework for delegation to an assistive user agent. In *AAAI 2005 Fall Symposium Series (to appear)*, Arlington, VA, Nov. 2005.
13. B. Peintner and M. E. Pollack. Low-cost addition of preferences to DTPs and TCSPs. In *Proc. of AAAI-04*, pages 723–728, San Jose, CA, 2004.

14. M. S. Pini, F. Rossi, and K. B. Venable. Possibility theory for reasoning about uncertain soft constraints. In *Proc. of ECSQARU 2005*, Barcelona, Spain, July 2005.
15. F. Rossi, K. B. Venable, and N. Yorke-Smith. Controllability of soft temporal constraint problems. In *Proc. of CP'04*, LNCS 3258, pages 588–603, Toronto, Canada, Sept. 2004.
16. SRI International. CALO: Cognitive Assistant that Learns and Organizes. `www.ai.sri.com/project/CALO`, Mar. 2005.
17. K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.
18. I. Tsamardinos. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence, Second Hellenic Conf. on AI (SETN 2002)*, LNCS 2308, pages 97–108. Springer, 2002.
19. I. Tsamardinos, M. E. Pollack, and S. Ramakrishnan. Assessing the probability of legal execution of plans with temporal uncertainty. In *Proc. of ICAPS'03 Workshop on Planning Under Uncertainty and Incomplete Information*, Trento, Italy, June 2003.
20. I. Tsamardinos, T. Vidal, and M. E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.
21. K. B. Venable and N. Yorke-Smith. Disjunctive temporal planning with uncertainty. In *Proc. of IJCAI'05*, pages 1385–1386, Edinburgh, UK, Aug. 2005.
22. T. Vidal and J. Bidot. Dynamic sequencing of tasks in simple temporal networks with uncertainty. In *Proc. of CP'01 Workshop on Constraints and Uncertainty*, pages 39–48, Paphos, Cyprus, Dec. 2001.
23. T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: From consistency to controllabilities. *JETAI*, 11(1):23–45, 1999.

The Basic Ingredients of a Constraint-based Framework for Decision-making under Uncertainty

Gérard Verfaillie[1] and Cédric Pralet[2]

[1] ONERA/DCSD, Toulouse, France, Gerard.Verfaillie@onera.fr
[2] LAAS-CNRS & INRA/BIA, Toulouse, France, Cedric.Pralet@laas.fr

Abstract. This main thesis of this position paper is that classical constraint reasoning tools are not suited to represent and to solve satisfactorily decision-making problems under uncertainty and that new frameworks must be built for this purpose. Such frameworks should maintain the basic ingredients of constraint-based reasoning, that are the notions of *variable*, *domain*, and *local relation* between variables. They should however establish a clear distinction, on the one hand, between *controllable* (decision) and *uncontrollable* (environment) variables and, on the other hand, between *feasibility*, *uncertainty*, and *utility* relations.

1 Many various settings

Decision-making under change and uncertainty is a large umbrella under which one can find many decision-making problems with various settings and objectives.

These settings differ in the nature of the changes that may occur : changes in the state of the system to manage, in the state of its environment, or in the objectives that are assigned to it by users.

They differ in the knowledge the decision module may have of the current state of the system and of its environment: partial or complete. They differ in the knowledge the decision module may have of the future changes in the state of the system and of its environment and in the objectives that will be assigned to it: partial or complete. A partial knowledge may have itself different forms: only a frontier between what is possible and what is not, a possibility distribution with several levels of possibility, a probability distribution ... They differ also in the way this knowledge may evolve: no possible change, natural changes without any action, changes thanks to observation or communication actions ...

These settings differ in the possible decision times : only one decision time (off-line planning), several decision times (successive off-line plannings), decision possible at any time (on-line planning). They differ also in the decisions that are possible at these times : any decision, only some types of decision ... They differ in the possible cost of modifying previously made decisions : modifications only possible for some types of decision, modification cost according to the type of decision and to the extent of the changes ... They differ also in the computing

resources available at these times to make decisions: cpu, memory, time ... They differ in the deadlines of decision-making : hard deadlines, soft deadlines, no deadline, possible trade-off between decision delivery time and decision quality ...

These settings finally differ in the type of objective which is assigned to the system : to achieve a specified goal, to guarantee a specified service, to optimize a service ... In case of service, they differ in the horizon over which it must be guaranteed or optimized : finite or infinite.

Combining all these features gives rise to completely different problems. For example, (i) the control of a robot in a partially known environment, only accessible via biased captors, in order to have the guarantee that it will be at a given location by a given deadline, (ii) the centralized management of a fleet of delivery vehicles in order to satisfy as well as possible client requests which arrive at any time in the distribution center, (iii) the management from a ground mission center of a constellation of observation satellites which are each only accessible during limited visibility windows, taking into account uncertainty at decision time about the cloud cover of the ground areas to observe, (iv) the management of the deployment of a constellation of satellites taking into account uncertainties about failures of new satellite launches and of previously launched operational satellites.

2 Numerous existing frameworks

The consequence of such a diversity is that it is difficult for a framework to represent and handle satisfactorily all the features of all the possible problems.

For example, *Markov decision processes* (MDP [1]) need to explicit the transition probability and the local reward for each triple state-action-state, that is the complete system transition model. This does not apply in settings such as (ii) where no model of the client request arrivals is available, unless one accepts to learn it progressively. The same kind of difficulty occurs with *Influence diagrams* [2], in spite of a more compact variable-based representation of states, actions, and transitions, inherited from *Bayesian networks* [3].

Competitive analysis [4] does not need such a model. It considers a decision made with partial knowledge and the worst impact of this decision on the system output, taking into account all the possible scenarios. Consequently, it does not suit many settings such as (iii) or (iv), where available probabilistic information can dramatically improve decision quality.

Stochastic programming [5] is an attempt to extend mathematical programming to a setting where values of some parameters in the equations are uncertain. Consequently, it allows large off-line planning problems with uncertain data to be handled, but does not suit settings such as (i), which are characterised by a reactive decision/action/information-acquisition loop.

In the classical AI *Planning* domain [6], several standard planning problems under uncertainty have been defined, such as (a) *Conformant planning* where

one searches for a plan that guarantees to achieve the goal in spite of state unobservability, (b) *Probabilistic Planning* where one searches for a plan that maximizes the probability of goal achievement, or (c) *Conditional planning* where one builds a flexible plan with alternative branches associated with at least the most important or probable events. But, off-line approaches, such as (a) or (b), do not suit on-line decision settings such as (i) or (ii), and on-line ones, such as (c), are quickly limited by the exponential number of alternatives to consider.

In the *Constraint* domain, one can observe preliminary attempts to consider uncertainty. First, it must be observed that the basic CSP model allows uncertainty alone to be represented and handled, because it allows possible and impossible states to be distinguished. This allows for example pure diagnosis applications to be dealt with by constraint programming tools (see for example [7]). Then, several extensions have been defined such as (a) *Mixed CSP* [8], where controllable and uncontrollable variables are distinguished and one usually searches for an assignment of the controllable variables that is consistent with any assignment of the uncontrollable ones, (b) *One stage stochastic CSP* [9], where one assumes each uncontrollable variable to be an independent random variable with an associated probability distribution and one searches for an assignment of the controllable variables whose probability of consistency is maximum, (c) *Probabilistic CSP* [10], where a probability of existence in the real world is associated with each constraint and one searches for a complete assignment whose probability of consistency is maximum, (d) *Branching CSP* [11], where a probability of addition to the problem is associated with each variable and one searches for an assignment of the current variables that maximizes the expected value of its extension to the future variables, (e) *Quantified CSP* or *Multi stage stochastic CSP* [12,9], where a request is built by freely alternating controllable existentially quantified variables and uncontrollable universally or randomly quantified ones. Note that other approaches, maybe the most usual in real-world applications, deal heuristically (without any model) with uncertainty by trying and producing decision variable assignments that are likely to be solutions in the real world (see for example [13]). Finally, whereas all these approaches aim at producing *robust* assignments, other ones aim at producing *flexible* ones. This is the case when one uses algorithms able to produce quickly a new solution in case of change in the problem definition which invalidates the previous solution, with an as small as possible distance between new and previous solutions (see for example [14,15]). This is also the case when one searches for solutions, referred to as *super-solutions*, that are presumably easy to repair [16].

But each of these extensions seems to have its limited application area, and none of them can claim to be the definitive answer. So, a generic constraint-based framework, sufficiently flexible to deal with all the possible features of uncertainty, remains to be defined. The definition of such a framework is not the aim of this paper. For a complete proposal, one can look at [17]. We can however list the basic ingredients about them there is no too strong contest.

3 The basic ingredients of a new framework

First, inherited from the CSP framework, we have the notions of *variable* and *domain*. But, whereas all the CSP variables are assumed to be controllable decision variables, we must now distinguish *controllable* (decision) variables and *uncontrollable* (environment) ones. About the former, we must decide upon their values. About the latter, we have only knowledge about their possible values.

Then, inherited from the CSP framework too, we have the notion of *local relation* between a limited number of variables. This is the basis of the *local consistency algorithms* [18] which made the success of constraint programming. This notion has been already extended from *hard* relations, which are satisfied or not, to *soft* ones, which associate a weight with each possible assignment of the variables they link [19] and can be seen as simple local functions. But, we must now distinguish three kinds of relation: (a) *feasibility* relations which express limitations on the decision variables as functions of other variables (decision or environment ones), (b) *uncertainty* relations which express beliefs on the environment variables as functions of other variables (decision or environment ones), and (c) *utility* relations which express user requirements or preferences on the decision and environment variables. Relations of type (a) are classical hard constraints, but must be normalized to guarantee that a decision is always possible (which may be to do nothing or to do something unacceptable). Relations of type (b) may be hard. In this case, they distinguish possible and impossible values of the environment variables. They may be also soft. In this case, they express conditional belief distributions (for example, possibility or probability distributions) over the domains of the environment variables. In both cases, they must be normalized to guarantee that a reaction of the environment is certain. Relations of type (c) may be hard. In this case, they express strong user requirements. They may be also soft. In this case, they express user soft preferences. In both cases and differently from feasibility and uncertainty relations, there is no normalization condition.

Finally, such a situation strongly differs from the one we are used with the CSP framework, where we deal with only one kind of variable, which are controllable decision ones, and only one kind of constraint, which express indistinguishably hard feasibility relations or hard utility ones (strong user requirements). The main novelty is thus that we have now to deal with two kinds of variables (decision and environment) and three kinds of local relations (feasibility, uncertainty, and utility). Because feasibility, uncertainty, and utility relations are not combined the same way (for example, feasibility relations may be combined with the logical $\wedge$ operator, uncertainty ones with $\times$ in case of probability, and utility ones with $+$ in case of numeric additive utility) and because decision and environment variables are generally not quantified the same way (for example, decision variables may be quantified with max and environment ones with $+$ in case of expected numeric additive utility), all the basic constraint reasoning tools (local consistency enforcing, tree search, local search ...) must be revisited to be adapted to a completely new framework.

References

1. Puterman, M.: Markov Decision Processes, Discrete Stochastic Dynamic Programming. John Wiley & Sons (1994)
2. Howard, R., Matheson, J.: Influence Diagrams. In: Readings on the Principles and Applications of Decision Analysis. Strategic Decisions Group, Menlo Park, CA, USA (1984) 721–762
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
5. Birge, J., Louveaux, F.: Introduction to Stochastic Programming. Springer Series in Operations Research (1997)
6. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
7. Yorke-Smith, N., Gervet, C.: Certainty Closure: A Framework for Reliable Constraint Reasoning with Uncertainty. In: Proc. of CP-03, Cork, Ireland (2003) 769–783
8. Fargier, H., Lang, J., Schiex, T.: Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge. In: Proc. of AAAI-96, Portland, OR, USA (1996) 175–180
9. Walsh, T.: Stochastic Constraint Programming. In: Proc. of ECAI-02, Lyon, France (2002)
10. Fargier, H., Lang, J.: Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach. In: Proc. of ECSQARU-93, Grenade, Spain (1993) 97–104
11. Fowler, D., Brown, K.: Branching Constraint Satisfaction Problems for Solutions Robust under Likely Changes. In: Proc. of CP-00, Singapore (2000) 500–504
12. Bordeaux, L., Montfroy, E.: Beyond NP: Arc-consistency for Quantified Constraints. In: Proc. of CP-02, Ithaca, New York, USA (2002) 371–386
13. Wallace, R., Freuder, E.: Stable Solutions for Dynamic Constraint Satisfaction Problems. In: Proc. of CP-98, Pisa, Italia (1998) 447–461
14. Minton, S., Johnston, M., Philips, A., Laird, P.: Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. Artificial Intelligence **58** (1992) 160–205
15. Verfaillie, G., Schiex, T.: Solution Reuse in Dynamic Constraint Satisfaction Problems. In: Proc. of AAAI-94, Seattle, WA, USA (1994) 307–312
16. Hébrard, E., Hnich, B., Walsh, T.: Super Solutions in Constraint Programming. In: Proc. of CP-AI-OR-04, Nice, France (2004) 157–172
17. Pralet, C., Verfaillie, G., Schiex, T.: Composite Graphical Models for Reasoning about Uncertainties, Feasibilities, and Utilities. In: Proc. of the CP-05 Workshop on "Preferences and Soft Constraints", Sitges, Spain (2005)
18. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
19. Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems : Hard and Easy Problems. In: Proc. of IJCAI-95, Montréal, Canada (1995) 631–637