Constraint Propagation in Planning and Scheduling

Claude Le Pape

CIFE TECHNICAL REPORT

Number 29

January, 1991

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

FOREWORD

Most of the work presented in the following report has been performed in the context of two projects aimed at controlling the actions of multiple agents in the same environment.

- **Indoor Automation With Many Mobile Robots.** The goal of this project is to control the operations of many mobile robots (several dozens) in an indoor environment (an office environment, a shop-floor, an airport, an hotel) in order to automate a variety of tasks. Typical tasks include transportation of objects (beverages, books, mail), operation of machines (copiers, vending machines), cleaning and maintenance. Different tasks may require different physical capabilities. Nevertheless, most of the tasks essentially involve mobility and transportation of relatively small objects.
- **Planning, Scheduling and Monitoring Actions of Multiple Agents on a Construction Site.** This project concerns the integration of various short-term planning, scheduling and execution monitoring techniques for multiple agents (robots and humans) working on a construction site. The case of a construction site introduces additional difficulties. For example, the geometry of a construction site continually changes. More sophisticated planning techniques (integrating temporal and geometrical reasoning) are consequently needed.

The current planning, scheduling and execution system integrates a collection of software components: a **task planning system** to derive plans made of "high-level" actions from a description of the tasks to be performed; a **task allocation system** to order and allocate tasks or actions to agents; a **motion planning system** to convert high-level actions into motion commands; and an **execution system** to monitor execution and react to unexpected events. These components are implemented in COMMON-LISP on a DEC 3100 workstation. The overall system is tested with the help of a simulation system designed to simulate actions of autonomous agents.

A preceding report presents the simulation system. Here, we show how we have used the simulation system to test and compare constraint-based planning and scheduling methods. The report starts with an introduction to constraint propagation, planning and scheduling and with theoretical considerations on constraint propagation (sections 1 and 2). Two applications of constraint propagation in the context of the projects above follow. Section 3 investigates situation-independent planning and section 4 discusses different task allocation methods involving constraint propagation. Then we present manufacturing applications involving similar issues. Section 5 presents previous experiments in the manufacturing scheduling domain and section 6 presents an architecture allowing a predictive scheduler and a reactive dispatcher to run in parallel and deal with environmental uncertainty in a consistent fashion. We close the report with prospective considerations on (a) software verification in the planning and scheduling domain and (b) the expression of control knowledge allowing to make the best use of constraint propagation (sections 7 and 8).

Constraint Propagation in Planning and Scheduling

Claude Le Pape
Robotics Laboratory
Department of Computer Science
Stanford University
Palo Alto CA 94305 USA

Abstract: Constraint propagation is a deductive activity performed by a constraint propagation system for a problem-solver. It enables the problem-solver to decompose a problem without neglecting interactions between subproblems, determine which subproblems are the most constrained and focus attention accordingly. Constraint propagation techniques are often used to solve planning and scheduling problems. Experiments reported in the past have shown the interest of these techniques in the planning and scheduling domain. In this report, we review various constraint propagation techniques and provide experimental results allowing to compare them on a variety of problems. We conclude that current constraint propagation systems do not enable a human problem-solver to make his (or her) own applications as efficient as possible without the help of a specialist. An interesting avenue of research is to make the adaptation of a generic constraint propagation system manageable by its users. Another is to provide the system with the ability to learn from its experience and adapt itself to the type of planning and scheduling problems it encounters.

Keywords: Constraint propagation, planning, scheduling, predictive problem-solving, reactive problem-solving, incremental problem-solving.

Acknowledgements: Most of this research has been funded by the Center for Integrated Facility Engineering (CIFE), the Center for Integrated Systems (CIS) and Digital Equipment Corporation (DEC). The development of the task planning and task allocation systems presented in sections 3 and 4 started in 1989 while the author was supported by ELF-Aquitaine. The scheduling system discussed in section 5 was developed in 1987 and 1988 by Anne Collinot, Gérard Pinoteau and the author at Laboratoires de Marcoussis. The first version of the execution monitoring system discussed in section 6 was designed by Karl Kempf, Naiping Keng and Stephen Smith at Intel Corporation. Section 6 reflects intensive discussions with them and with Malcolm Acock, Mark Drummond and Srinivas Narayanan. I would like to thank Jean-Claude Latombe for the chance I had to work in the Robotics Laboratory, and the members of the Robot Reasoning Group for interesting discussions on constraint propagation, planning and scheduling.

Contents

1 Introduction

"The *divide-and-conquer* strategy breaks large problems into smaller, more tractable subproblems. Ideally, the solution of a complete problem simply requires combining the solutions of independently-solved subproblems. In practice, subproblems often interact. When subproblems are only *nearly independent*, solving a complete problem involves coordinating the solutions of its subproblems ... This thesis demonstrates an approach to problem-solving, termed constraint posting, which combines the traditional constraint satisfaction ideas with hierarchical problem-solving. Constraint posting has three main operations: (1) formulating constraints, (2) propagating constraints, and (3) satisfying constraints. While these operations could be classified simply as inferences, they perform distinct and important roles in problem-solving — specialization, communication, and coordination respectively. Using constraint posting, a planning program (named MOLGEN) can combine under-constrained subproblems to form well-constrained larger problems. As constraints are propagated between subproblems, MOLGEN eliminates interfering solutions without backtracking."

Mark Stefik [137]

Since the publication of Stefik's thesis and related articles [137] [138], many researchers have applied constraint posting to planning and scheduling problems. Constraint posting allows to anticipate subproblem interactions and therefore to reduce search. In addition, the explicit statement and satisfaction of constraints enables the generation of more realistic plans and schedules. While rigid algorithms designed to solve particular problems cannot accommodate constraints ignored at design time, a constraint system allows the formulation of a great variety of constraints (in a system-dependent formal language). This happens to be very important in the case of factory scheduling since important features of scheduling problems vary from one shop to another [92]. It is even more important in the domain of civil engineering automation — domain in which distinct planning problems are often radically different.

Constraint posting techniques have allowed to solve difficult problems. Nevertheless, research is still needed to solve a lot of manufacturing and engineering problems. For example, the most successful constraint-based reactive scheduling systems (e.g. [13] [31] [131]) are able to correct disrupted factory schedules in a few seconds or less. This is almost satisfactory for small shop-floors with small event rates (typically producing mechanical parts). But this is much too slow for complex environments such as integrated circuit factories.

In this report, we discuss recent applications of constraint propagation techniques to various planning and scheduling problems. The goal is not to demonstrate the interest of these techniques. We already believe in this interest (and refer the skeptical reader to [149] which contains many impressive results). Instead we review various constraint propagation techniques and provide experimental results allowing to compare them on various planning and scheduling problems.

This section introduces the concepts of constraint propagation, planning and scheduling to the unfamiliar reader.[1] Section 2 presents theoretical foundations underlying the constraint propagation systems discussed in this report. Sections 3 to 6 present constraint-based planning and scheduling applications together with experimental results. Section 7 discusses envisioned applications of constraint propagation to software verification in the planning and scheduling domain. Section 8 summarizes the most important results and proposes directions for further research.

1.1 Constraint Propagation

Constraint propagation is a deductive activity performed by a constraint propagation system for a problem-solver or for several problem-solvers operating in parallel. It enables the problem-solver to decompose a problem without neglecting interactions between subproblems, determine which subproblems are the most constrained and focus attention accordingly. The constraint propagation system derives new constraints from existing ones. It also detects inconsistencies between several types of constraints, e.g. (in the planning and scheduling domain) goals, decisions, preferences, occurring and predicted events (figure 1).

Mathematically speaking, the main concept in constraint propagation is the concept of a variable "held constant" "with a conditional interpretation" [77] [87].

- "Held constant" means that a variable x stands for the same object throughout all the constraint statements (and throughout constraint propagation).
- "With a conditional interpretation" means that every statement containing a free variable x expresses a condition on x or — more precisely — on the object x stands for.

In other terms, a constraint set $C = \{P(x)\ P(y)\ Q(x\ y)\}$, where P and Q are predicates and x and y variables, is used when one wants to determine (or prove the existence of) two values v and w, such that $P(v)$, $P(w)$ and $Q(v\ w)$ hold.

[1]It does not constitute a review of the huge amount of research performed in the planning and scheduling domain. Recent reviews in planning include [143] [144] [64] and the first chapter of Rit's PhD thesis [114]. [8], [17] and [23] are the most significant books about scheduling. Each of them provides a good survey of traditional (operations research) scheduling models and techniques. Steffen [136] reviews the scheduling systems developed before 1986 with artificial intelligence techniques (including constraint propagation). Kempf [73] provides a more recent bibliography as well as an interesting viewpoint on the evolution of research in the "artificial intelligence and scheduling" domain. Burke [14] presents and criticizes the most well-known scheduling systems using artificial intelligence techniques. Collinot [28] presents both operations research techniques and artificial intelligence systems for scheduling — and establishes an interesting link between the two scheduling research fields. All of this background is not necessary to understand the present report. On the other hand, a little background in operations research [71], automated deduction [77] [115], graph theory [10] [59] [60] and complexity theory [51] is necessary.

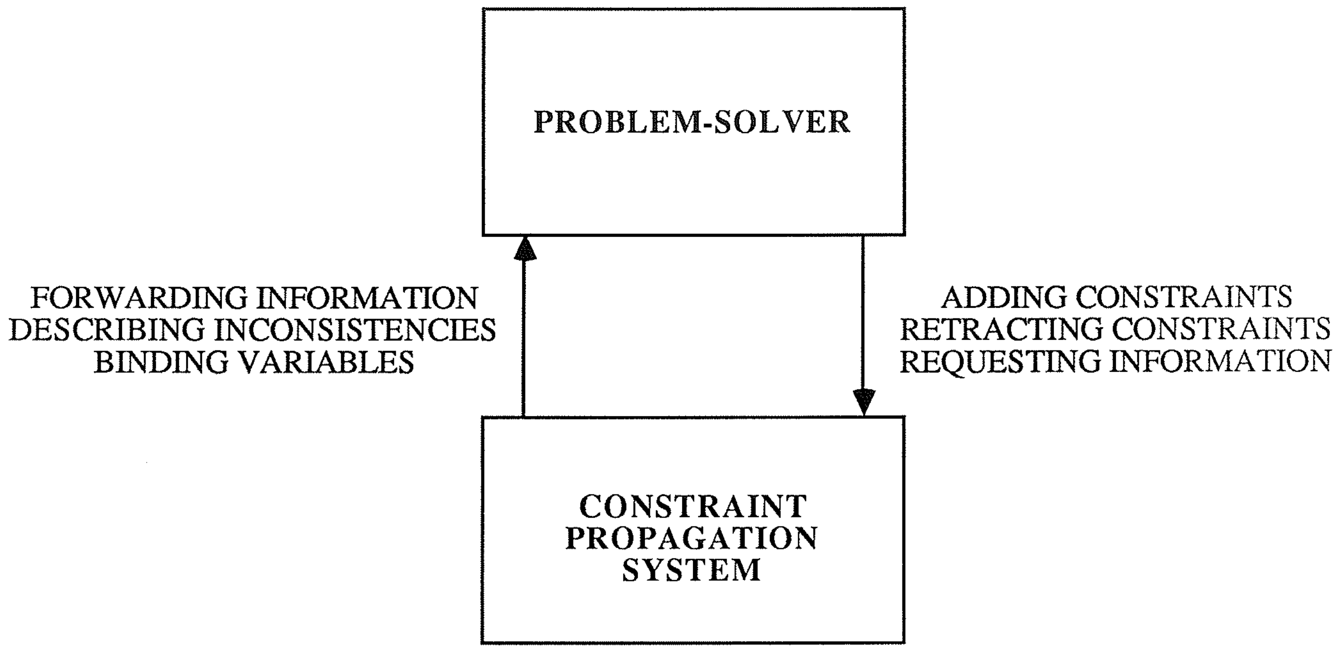

Figure 1: Constraint Propagation

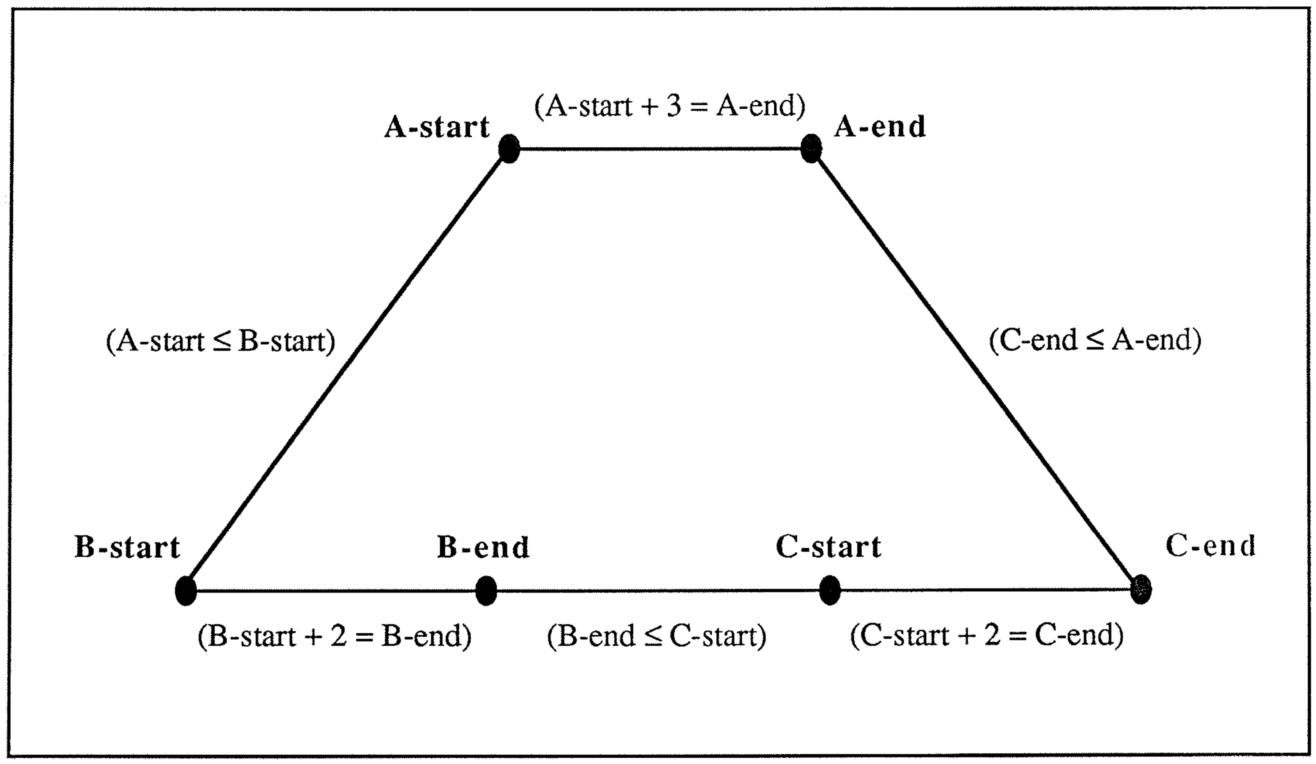

Figure 2: A Binary Constraint Network

Constraints provide a specification of admissible assignments of values to variables. Domain constraints such as $x \in \{2\ 4\ 8\ 16\}$ and $y \in \{2\ 4\ 8\ 16\ 32\ 64\}$ describe domains over which variables can vary. Variable relations such as $(x + 1) \geq (2 * y * y)$ define a subset of the cartesian product of these domains. Variables and constraints are often represented in an hypergraph whose vertices are the variables (with the associated domains) and whose hyperedges correspond to the variable relations. In particular, constraints involving two variables are often organized in a graph as in figure 2. When such a graph (hypergraph) is used as a representation for a constraint satisfaction problem, it is worth using its structure to choose constraint propagation steps. The determination of connected components and the detection of tree-like (hypertree-like) shapes within the graph are of particular interest. In particular, Shenoy and Shafer have designed a powerful constraint propagation algorithm for variables with finite domains and relations forming an hypertree [126]. This algorithm detects inconsistencies and determines values satisfying the constraints (when constraints are compatible) in $O(n * M^c)$, where M denotes the size of the largest domain, c the size of the largest hyperedge and n the number of constraints. It uses propagation operators similar to the operators of relational database management systems [37] and allows (with minor changes) to perform discrete optimization, compute marginals of probability distributions and find a most probable configuration of causes to explain a set of symptoms [125] [127].[2]

Different types of constraint propagation steps are useful depending on the domains and relations. For boolean variables, each constraint propagation step consists of deducing a formula in propositional logic. When domains are euclidean spaces and relations refer to euclidean properties, theorems of euclidean geometry are put into use. More generally, combinatorial and algebraic approaches to constraint propagation are distinguished (see [34]).

- A combinatorial approach consists of performing operations concerning the possible values of variables. For example, if x equals 1 and if y is constrained to be greater than x, only values greater than 1 are candidates for y.

- An algebraic approach consists of performing operations concerning relations between the unknown values of variables. For example, if x is constrained to be greater than y and if y is constrained to be greater than z, then x is constrained to be greater than z.

Combinatorial approaches are often applied in finite domains [126] [149] while algebraic approaches are often applied in infinite but homogeneous structured domains. In particular, algebraic approaches are often used to reason about time [4] [56] [79] [87] [88] [119] [150].

[2]Seidel [124] presents another (less general) algorithm which suits graphs with particular structural properties. It runs in $O(n * M^{f+1})$ where $f < n$ is an integer which depends on the graph and $(f + 1)$ equals c (the size of the largest hyperedge) for *some* linear hypertree cover of the graph. Arnborg, Corneil and Proskurowski [6] propose an algorithm in $O(v^{k+2})$ to determine whether a graph with v vertices is a partial k-tree and compute the embedding k-tree. Given the k-tree, one can construct an hypertree cover for the original graph with hyperedges of size $(k + 1)$ (in $O(v^{k+1})$) and then use Shenoy and Shafer's algorithm to (1) compute the corresponding variable relations (in $O(n * M^{k+1})$) and (2) solve the resulting problem (in $O(n * M^{k+1})$). Hence the combination of the two algorithms in [6] and [126] runs in $O(v^{k+2} + n * M^{k+1})$ for a partial k-tree.

When constraints are not compatible, it is useful to provide the user of the constraint propagation system (human or computer system) with a description of the detected conflicts. For example, such a description allows to perform "relevant" backtracking and to avoid recreating the conditions of a failure [12] [81] [82] [120] [134].[3] To provide such a description, the constraint propagation system must remember which constraints have been derived from other constraints. This leads to defining techniques similar to those used in truth maintenance systems [35] [36]. Conversely, truth maintenance problems have been mentioned as particular constraint satisfaction problems [34]. The two research domains (truth maintenance and constraint propagation) are consequently very close one to the other. In particular, the same complexity problem arises from the use of disjunctions in the two domains.

When disjunctive constraints (or constraints with an intrinsic disjunctive nature) are considered, the problem of determining whether a given set of constraints is consistent is NP-hard. Therefore, polynomial systems are (unless P = NP) unable to detect all possible contradictions between disjunctive constraints. Most constraint propagation systems perform a fixed amount of propagation: the trade-off between the anticipation of interactions and the amount of computational effort spent in constraint propagation is set once for all. More flexible systems allow to reduce or extend propagation with respect to a predetermined set of particular parameters (e.g. reference intervals in [4], levels of precision in [86]) or include a small programming language to control constraint propagation [28] [87] [149]. Results of recent experiments with such systems [30] [149] show that one should carefully consider the control of constraint propagation to improve the efficiency of the resolution of constraint satisfaction problems: the amount of constraint propagation that enables a problem-solver to be the most efficient varies with the problem-solver, with the application (e.g. in job-shop scheduling, the appropriate amount of propagation varies from one shop to the other) and with the problem-solving context (e.g. in case of urgency, propagation can be restricted to constraints relating to imminent manufacturing operations). This gives particular impor-

[3]There are a few terminological problems in the literature on "relevant" backtracking. In particular, the distinction between *chronological* and *dependency-directed* backtracking varies from an author to the other: removing decisions in reverse chronological order until one responsible for the failure is removed is *chronological* for some authors and *dependency-directed* for others (it is in fact both *chronological* and *dependency-directed*). In this report, we use the following terms:

- *naive chronological* for removing the most recent decision and restarting the search from there (this is "naive" since the most recent decision is not always responsible for the failure);
- *intelligent chronological* for removing decisions in reverse chronological order until one responsible for the failure is removed;
- *selective* for removing the most recent decision among those responsible for the failure;
- *chronological* for *naive chronological* or *intelligent chronological* (when the distinction is useless);
- *dependency-directed* for *intelligent chronological* or *selective* (when the distinction is useless).

As mentioned in [99], selective backtracking has the effect of "jumping" in the search tree from one branch to another. Because it does not remove decisions made *after* the decisions responsible for the failure, selective backtracking does not always *backtrack* to a point in the search space that has been visited before.

tance to two research areas mentioned in [66]: (1) the development and characterization of intrinsically flexible inference strategies and (2) the mastery of techniques and computational architectures for efficient control.

1.2 Planning and Scheduling

This section presents the planning and scheduling research areas to the unfamiliar reader. It introduces concepts needed to understand the following sections. We present planning and scheduling problems in section 1.2.1 and then discuss more complex issues: predictive and reactive planning and scheduling (section 1.2.2), planning and scheduling with uncertainty (section 1.2.3), distributed and centralized planning and scheduling (section 1.2.4).

1.2.1 Planning and Scheduling Problems

A plan is a program of actions that can be carried out to achieve goals. **Planning** means to create plans [139]. In this report, we are principally interested in **task planning**, i.e. in deriving programs of high-level actions (e.g. "go to position P") from a description of the goals to be achieved, as opposed to **motion planning**, which consists of converting these high-level actions into motion commands. **Scheduling** consists of ordering and distributing plans or single actions among agents. The distinction between planning and scheduling is fuzzy: a plan and a schedule are both programs of actions enabling the achievement of goals. Formal definitions of "planning" (e.g. [102]) tend to accept "scheduling" as a particular case. In this report, we will use the term "planning" to emphasize the *design* of a set of actions which will achieve the goals. We will use the term "scheduling" when the problem is to determine *who* will execute *which* actions and *when*. This does not mean there is no temporal concern in planning. In most cases, the actions of a plan must be executed in some order for the plan to succeed. However, ordering decisions are not always necessary at planning time. Following Fox and Kempf [48], we differentiate planning and scheduling as follows:

- **Planning**: given an initial world, a goal world, and a set of operators (a task and a set of facility capabilities), select a set of operators which will achieve the goal, and generate a minimal set of ordering constraints on operator application.
- **Scheduling**: given a set of operators and minimal ordering constraints (a plan) and detailed knowledge of the execution environment (a set of facility availabilities), enforce further ordering constraints on operator application to achieve robust and time-efficient execution of the task.

Task planning and scheduling activities are in most cases triggered by the communication of **orders**. Each order is submitted to an agent by another agent referred to as the **client**. The original description of an order consists of a set of **goals** and a set of **temporal constraints** concerning the goals. Goals are statements the client wants to be true at some points in time. Constraints indicate how these points in time are (ideally) related. Three broad classes of constraints are distinguished. **Restrictions** delineate the space of admissible solutions to the considered problem while **preferences** are constraints whose satisfaction may be compromised if necessary. The third class, referred to as **decisions** (or as "conflict-avoidance preferences" [86]), is composed of constraints induced as a result of previous planning and scheduling decisions. **Penalties** are associated with preferences to measure the importance of each preference violation. For example, figure 3 presents a constraint to achieve a goal G before a given due-date and a description of the penalty as a function of the delay. Penalties are often combined with respect to **weighted indexing schemes**. The basic approach is to attach a relative weight to each preference. For each problem solution, this weighting allows to construct a weighted average of the penalties (index). This enables the comparison and the choice of alternative solutions.[4][5]

Planning and scheduling problems are — except in a few cases in which simple decision rules apply (e.g. see [70] [68]) — extremely complex.[6] Chapman [21] even mentions that any Turing machine with its input can be encoded as a planning problem: the problem of proving the existence of a plan to achieve a set of goals given an initial situation and a set of operators in Chapman's language is not decidable. In section 3, we will nevertheless see that constraint

[4]One can also construct several indexes and compare them to thresholds in order to decide whether a solution is acceptable or not. See [5] for a good example in the domain of organization monitoring.

[5]The relative value of the index for a set of alternatives changes depending upon the set of constants employed to normalize the values of the penalties applied. In [46], Field, De Neufville and Clark provide a striking example in the domain of material selection and criticize weighted indexing techniques on this account. Their approach is to interview experts and construct global penalty functions which are not always linear combinations of individual penalties.

[6]Let us show that the planning problem for a conjunctive goal in propositional logic and with one precondition and one effect per operator is NP-hard ([51] and [84] provide NP-hardness results in scheduling). Let us consider a set of clauses $\{C_1 \ldots C_n\}$ where each clause C_i equals $\{L_{i1} \ldots L_{in_i}\}$. Let $P_+(C_i)$ denote the set of positive propositions which appear in C_i and $P_-(C_i)$ denote the set of positive propositions the negation of which appear in C_i (e.g. if C_i equals $\{A\ (NOT\ B)\}$, $P_+(C_i) = \{A\}$ and $P_-(C_i) = \{B\}$). Let P denote the union of all the $P_+(C_i)$ and $P_-(C_i)$. P is a finite set $\{P_1 \ldots P_p\}$. We define a planning problem as follows:

- $(2p+n)$ boolean variables describe a planning situation: $TRUE_k$ $(1 \leq k \leq p)$; $FALSE_k$ $(1 \leq k \leq p)$; and C_i $(1 \leq i \leq n)$.
- $TRUE_k$ $(1 \leq k \leq p)$ is false in the initial situation. $FALSE_k$ $(1 \leq k \leq p)$ is false in the initial situation. C_i $(1 \leq i \leq n)$ is false in the initial situation.
- The goal is $(C_1\ AND \ldots AND\ C_n)$.
- There are four kinds of operators: T_k $(1 \leq k \leq p)$; F_k $(1 \leq k \leq p)$; C_{+ik} $(1 \leq i \leq n$ and $P_k \in P_+(C_i))$; C_{-ik} $(1 \leq i \leq n$ and $P_k \in P_-(C_i))$. The precondition of T_k is $(NOT\ FALSE_k)$ and the effect of T_k is $TRUE_k$. The precondition of F_k is $(NOT\ TRUE_k)$ and the effect of F_k is $FALSE_k$. The precondition of C_{+ik} is $TRUE_k$ and the effect of C_{+ik} is C_i. The precondition of C_{-ik} is $FALSE_k$ and the effect of C_{-ik} is C_i.

The planning problem derives from the set of clauses in polynomial time and admits a solution if and only if the clauses are satisfiable. Hence planning is NP-hard since the satisfiability problem is NP-complete [51].

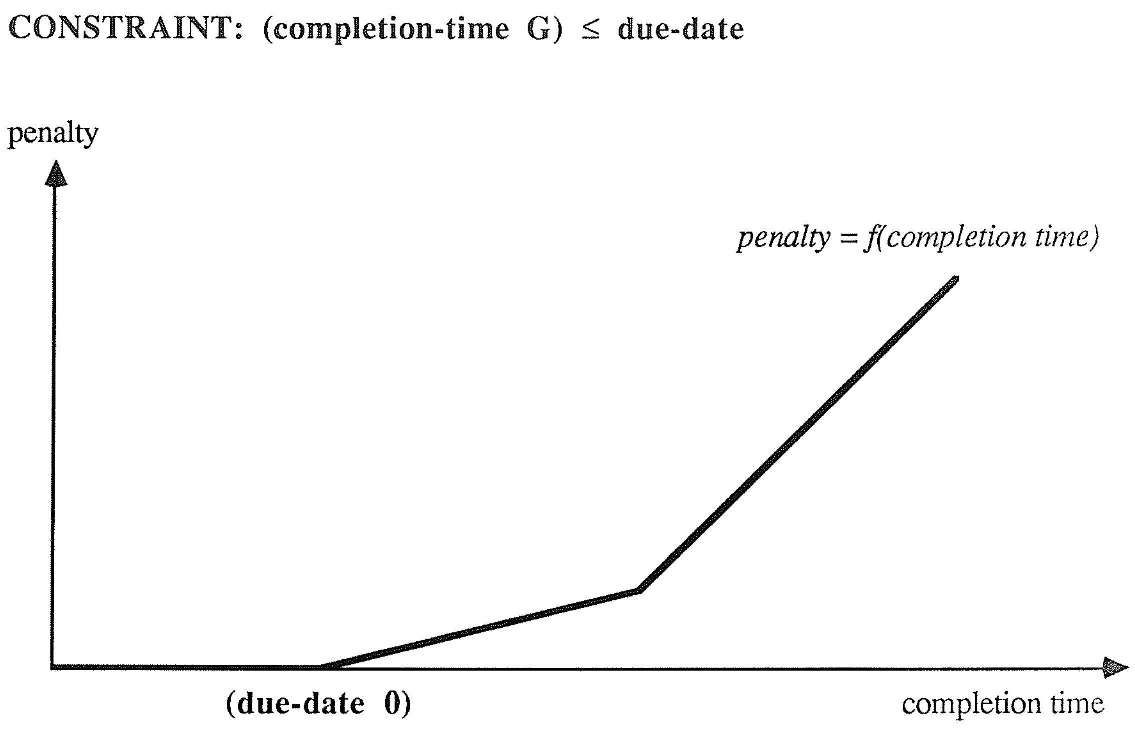

Figure 3: A Due-Date Constraint

propagation techniques allow to solve practical planning problems. In addition, a planning agent can often generalize plan structures and store them for subsequent use [47]: only parts of the optimization process must run for each goal instance. The situation is more difficult with respect to scheduling. The number of alternative solutions to a scheduling problem is huge and optimal solutions to similar scheduling problems are often radically different. Therefore, a number of researchers are devising techniques allowing the construction of satisfactory schedules (whatever that means) rather than optimal ones. The problem of proving the existence of a schedule which satisfies a set of temporal constraints is NP-hard [84] and the problem of proving whether a scheduling decision is consistent with a set of compatible constraints is NP-hard [87]. Complete constraint propagation for scheduling problems is therefore not practicable for complexity reasons.

Planning and scheduling problems are often solved one after the other. Given a set of goals, plans are created to achieve these goals. Complete specifications of actions are not needed at that time. For example, a plan for the task "provide John with a photocopy of this book" does not need to specify which photocopier to use. Then the scheduling problem includes distributing actions to agents and ordering actions that agents cannot execute in parallel. This separation between planning and scheduling simplifies coordination in complex environments (e.g. construction sites, factory floors, office environments). However, there are cases in which planning and scheduling problems must be solved altogether to insure the most efficient execution. Let us consider the following example:

> Mary orders a copy of David's thesis. There is one photocopier automatically operated by a fixed robot. Mary knows that John has a copy of David's thesis

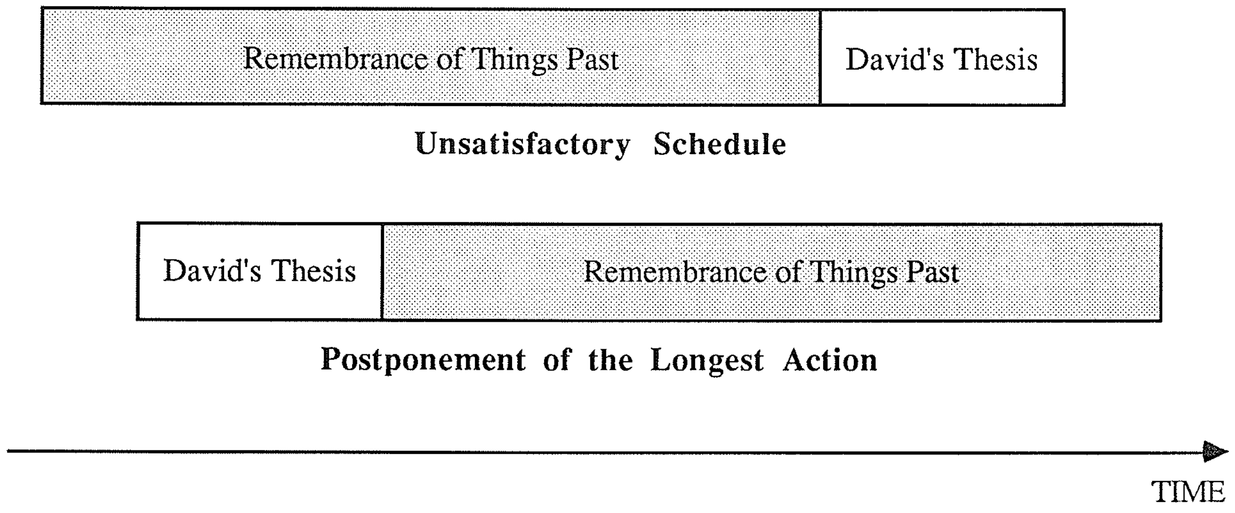
Remembrance of Things Past
David's Thesis
Unsatisfactory Schedule
David's Thesis
Remembrance of Things Past
Postponement of the Longest Action
TIME

Figure 4: Postponing Actions

and provides the office automation system with this information. John does not need his copy right now. But he will need it in the far future.

The best plan is to choose a mobile robot to (1) go to John's office, (2) get the thesis from John, (3) go to the photocopier, (4) get a copy of the thesis from the robot operating the photocopier, (5) bring the new copy to Mary and (6) bring the original back to John. However, if another mobile robot is waiting for copies of huge books, action (4) cannot be scheduled within a reasonable delay. In such a case, the best solution is to (1) go to John's office, (2) get the thesis from John and (3) bring it to Mary (in addition, we would like to warn Mary that John will need his copy in the future). To handle such cases, one can either alternate planning and scheduling steps or keep options open during planning.

The example presented above illustrates an important feature of planning and scheduling problems: plans interact. The best plan to provide Mary with a copy of David's thesis cannot be satisfactorily scheduled because a plan to provide X with copies of huge books is executed. A good way of dealing with interactions consists in combining plans, either prior to or during their execution. In many cases, scheduling actually consists in combining plans. If the fixed robot operating the photocopier knows in advance that a mobile robot will soon arrive with David's thesis, it may decide to postpone the reproduction of the huge books (figure 4). Postponing a long action is a very common scheduling decision, even though there is a risk of indefinitely postponing the longest actions. If the reproduction of the huge books is already started when the fixed robot receives information about David's thesis, it may accept to interrupt its actions to make a copy of David's thesis (figure 5). Such an interruption is referred to as "preemption". Depending on the environment, preemption is more or less time-consuming. In some cases, it is infeasible.

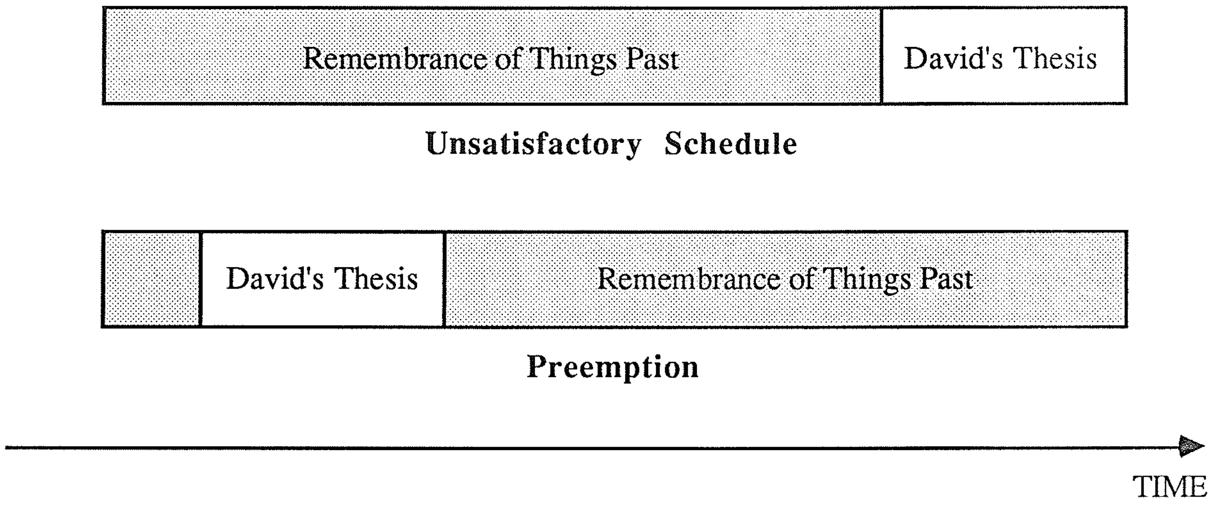

Figure 5: Preemption

Combining plans also means building a global plan from individual plans (each of which corresponds to the achievement of a task). Many interactions can be considered this way. However, planning decisions are not easy to make. For example, proposals have been made to detect situations in which an action contributes to the satisfaction of several goals [152] and to group actions requiring the use of the same resource [145]: to make two sets of transparencies for office-mates, a good plan is to (1) carry the two sets of originals to the same copier (one action), (2) make the two sets of transparencies one after the other (grouped actions) and (3) carry all the originals and transparencies back (one action). However, assigning different resources to unordered actions (e.g. one copier for each set of transparencies) enables a parallel execution of these actions and is in some cases the most time-effective way to achieve the goals [153].

1.2.2 Predictive and Reactive Planning and Scheduling

Unexpected events often occur during plan execution and prevent the completion of plans. Consequently, **predictive** and **reactive** planning and scheduling are distinguished:

- Predictive planning and scheduling consists in building and combining plans to be executed in the future.
- Reactive planning and scheduling consists in making decisions in real-time with respect to the actual state of the world. This does not mean every decision is made on-line, but previous decisions are confronted with unexpected events in order to detect (and react to) arising conflicts and opportunities.

This distinction is sometimes confused with the distinction between **goal-directed** and **data-directed** reasoning. Predictive systems are emphasized as determining how to achieve a set of goals and reactive systems as staying alert to incoming events. But in fact both predictive and reactive systems can make use of both goal-directed and data-directed rationalities:

- In a reactive system, unexpected events trigger a reaction. But the reaction often consists in creating a program of actions to achieve goals from the unforeseen current situation.
- Conversely, predictive systems often consider characteristics of initial situations as opportunities. For example, the SOJA predictive scheduling system [85] considers the fact that some tool T is set on some machine M as an opportunity: it allows to perform manufacturing operations requiring the use of both T and M without performing the setup of T on M.

In this report, a system is considered "reactive" if it is sensitive to the evolution (and not only to a given snapshot) of its environment. It is important to notice that the pertinence of a reaction is difficult to evaluate without engaging in complex discussions. When you show an average-size book B after a small book A to a human being, he (she) is more likely to bet that B is more than 300 pages long. When you show the same book B after a large book C, he (she) is more likely to bet that B contains less than 300 pages. One can consider such a fact as a sign of irrational behavior (especially if the human subject manipulates books all the time) or argue that similar reactions are apropos in most real-life cases.

The distinction between predictive and reactive problem-solving is often discussed, especially in the scheduling domain. There are several reasons for this:

- Predictive and reactive problem-solving can often make use of the same techniques. For example, the cognitive model of planning proposed in [63] is applicable to both predictive and reactive planning problems.
- Predictive and reactive problem-solving can often make use of common pieces of knowledge (e.g. preferences, indexing schemes, heuristics) and it is important that they do so. Otherwise a reactive scheduling system would make a lot of modifications to a schedule just because it would not consider it as a good schedule with respect to performance criteria.
- Reactive methods are used in predictive contexts. Indeed, the availability of reactive methods allows to build a schedule without worrying about all the constraints each time a decision is made. If some decisions happen to be inconsistent one with the other or if some scheduling preferences are violated to a too important extent, reactive methods can be used to repair the schedule.
- Predictive methods are used in reactive contexts. When a severe accumulation of unexpected events has forced important changes to an original schedule, it is worth generating a whole new schedule. Even if it is not suitable to emphasize optimization at the expense of responsiveness, predictive methods allow to perform this re-scheduling.

These points suggest the integration of predictive and reactive components in the same system and the use of the same "solution maintenance component" [89] to maintain plan or schedule descriptions as decisions are made and as unexpected events occur. Burke and Prosser [13] make a stronger statement. They argue that if we consider scheduling as search for one and only one solution satisfying some goals, then scheduling systems can be built without making any distinction between predictive and reactive scheduling. Such a scheduling system is presented in [13]. It consists of a society of agents, each agent being responsible to schedule one resource. Constraint propagation and truth maintenance techniques are used to maintain a consistent global hypothesis. When a conflict occurs, the agent with the smallest priority among the agents involved in the conflict uses dependency-directed backtracking to modify the schedule of the associated resource (if it cannot do it, it creates a new conflict in which it is not involved). There is no need to distinguish predictive and reactive contexts: the same dependency-directed backtracking algorithm can be applied in both cases.[7] Another method which applies in both cases is Liu's "scheduling via reinforcement" [95], which allows both the construction of a detailed predictive schedule from a rough capacity plan and the reactive revision of a disrupted schedule.

Still there is an important difference between predictive and reactive activities. Predictive activities are usually performed under no significant time stress while reactive activities are subjected to real-time constraints. Reactive systems cannot spend as much computational time as predictive systems to ensure the global quality of the solution. In some domains, **hard real-time constraints** indicate that reactive decisions must be made within strict delays. Rather than being fast, the most important property of reactive systems in such domains is the guarantee that they will produce responses on time [135]. In other domains, **soft real-time constraints** indicate that the utility of a given decision varies not only with the quality of the decision per se, but also with the time at which the decision is made. In some cases, optimizing a schedule may take a lot of time. Executing a sub-optimal schedule early is better than twiddling thumbs until an optimal solution is found.

1.2.3 Planning and Scheduling with Uncertainty

The need for reactive planning and scheduling arises from the existence of uncertainty in the considered environment. In this section, we discuss the use of knowledge about the existing uncertainty to better organize predictive and reactive activities: (1) to make better predictive and reactive decisions and (2) to balance predictive and reactive reasoning.

[7]This is often theoretically possible with dependency-directed backtracking algorithms, but not practicable for complexity reasons. For example, the predictive ordering component of the SONIA scheduling system [31] could be used to react when an unexpected event occurs, but it would be difficult to prevent subsequent explosions of search. Similarly, Descotte and Latombe's algorithm to make compromises among antagonist constraints [39] [40] [41] allows to add new constraints at any time. It backtracks when necessary, in an uncontrollable fashion.

Knowledge about the existing uncertainty allows to make better decisions. For example, one can use heuristics referring to the amount of uncertainty in each action to prioritize actions in time. However, the design of appropriate heuristics is a task to perform with care. Consider n independent actions and m identical robots ($1 < m < n$) able to execute each action: if the performance criterion is the average action completion time, a good heuristic is to postpone actions with unknown maximal durations in order to reduce the potential effects of a long execution; if the performance criterion is the maximal action completion time, a good heuristic is to execute the actions with unknown maximal durations first.

Another way of using knowledge about the existing uncertainty consists in applying more theoretical (and more time-consuming) methods based on decision theory [151] [67] and/or fuzzy logic and arithmetic [76]. Probabilistic or pseudo-probabilistic constraint propagation techniques (e.g. [11] [121]) are of high practical interest for this purpose.[8] Generally speaking, decision theory provides a framework to make planning and scheduling decisions with respect to knowledge characterizing the environmental uncertainty. The theoretical idea is very simple: the optimization of a function f (which combines various planning and scheduling preferences) is replaced with the optimization of the *expected value* of f given probabilistic knowledge on the possible consequences of contemplated actions and unpredictable events. However, the practical application of this simple idea poses complex problems:

- One is the acquisition of probabilistic knowledge. In most cases, the user of a planning and scheduling system does not know the exact probabilistic laws that the problem variables follow. Experimentation on a medium scale allows to approximate these laws, but the appropriateness of the approximation depends on a collection of hypotheses (e.g. representativeness) the satisfaction of which is not obvious. In addition, experimentation results often suggest changes which (once made) invalidate the results. For example, results showing that a particular machine often breaks down suggest the identification and the elimination of the most important breakdown causes. Then new experiments must be made to update the statistics. The "subjectivist" theory of decision analysis (see [83]) allows to do without experimentation when experimentation is too expensive (or when the problem is to decide whether a program of experiments is too expensive for the information it could provide). Once again the idea is simple: replace statistics with "subjective probabilities" reflecting the beliefs of an expert. The determination of the subjective probabilities requires some interaction with the expert. This interaction can be made less expensive than experimentation involving the actual environment. However, the precision of the results depends on the interaction process. Chu, Moskowitz and Wong [22] suggest that manipulating bounds on subjective prob-

[8]Curiously enough, most of these techniques have not been designed for uncertain reasoning. For example, the goal of Sadeh and Fox [121] is to use probabilistic distributions reflecting scheduling preferences to focus the attention of an incremental scheduler (see [122]) or of several schedulers operating in parallel (see [142]). As mentioned in [11], the Project Evaluation & Review Technique (PERT) for project scheduling uses probabilistic time estimates in order to cope with the existing uncertainty. But to our knowledge no exact probabilistic computation is made: PERT provides *conservative* results when required to do so.

abilities is more adequate than striving to get precise values. The system proposed in [22] requires the user to express strong preferences between a few problem solutions. From these preferences, it deduces constraints on subjective probabilities and eliminates solutions which (according to these constraints) cannot be optimal. Fagin and Halpern [45] also suggest the manipulation of bounds and define a semantic for them in perfect concordance with probability theory. This does not eliminate the acquisition problem, but enables the design of more reliable acquisition processes.

- Another complex problem is the combination of probabilistic pieces of information. Halpern [62] provides a collection of results showing that — under the two most common approaches to giving semantics to first-order logic with probabilities (i.e. probabilities on a domain or probabilities on a set of possible worlds) — there is no complete axiomatization and a fortiori no procedure to determine whether a first-order formula involving probabilities follows from a set of premises. There are three "solutions" to this theoretical problem: the first is to consider incomplete but sufficiently rich axiomatizations allowing to carry out "a great deal" [62] of interesting probabilistic reasoning; the second is to define complete axiomatizations for sub-languages of first-order probabilistic logic; the third is to define axiomatizations which do not allow the usual interpretations of probabilistic statements, but enable the resolution of practical problems. The choice among these three solutions is in most cases context-dependent and subjective. The first solution requires the acceptance of incompleteness. The second solution requires the definition of an appropriate sub-language for a definite class of applications. The third solution requires the rejection of probabilistic reasoning as a basis for decision-making (and its replacement with other models such as fuzzy logic and arithmetic — see [76] for a motivating example). The resulting "quasi-probabilistic" reasoning systems are in general not well-founded on analytical grounds and need a scrupulous validation on empirical grounds [123]: one must gather experimental results to make sure that the system "performs well".

Another (distinct) problem that arises in the planning and scheduling domain (and in the presence of uncertainty) is to balance predictive and reactive reasoning [74]. In some environments, detailed predictive decisions are not useful. For example, the use of the KANBAN system to replace parts as they are being used in a factory [49] makes the generation of a detailed factory schedule useless. The only schedule required is an assembly schedule which implies a consistent flow of parts through the work centers of the factory. When such an assembly schedule exists, no schedule for the production of parts is needed (and parts are nevertheless almost always available on time). In other environments (for example, when a work center involved in the production of parts is the bottleneck of the factory and its capacity cannot be increased at an acceptable cost), the generation of a detailed schedule allows to use available resources more efficiently:

- to prepare tools, make transportations and perform setups in advance;

- to reduce the idle time of the bottlenecks and hence augment the global factory throughput [58].

However, the uncertainty in the environment suggests the maintenance of many ordering possibilities allowing a reactive system to cope easily with unexpected events during execution. An extreme solution is to predictively assess all possible execution-time contingencies and plan responses to them. Thus, agents executing plans are provided with conditional plans prescribing actions in all possible situations. This solution is difficult to implement in real domains because there are too many non-similar possible situations to determine in advance how to react to all of them. Another extreme solution is to construct plans and schedules allowing the achievement of goals in the absence of surprises and re-plan from scratch when unexpected events occur. This works well in environments where unexpected events seldom occur. But in most cases the best solution is to get a plan which covers various (the most likely) possible situations and revise it (but not completely re-construct it) in response to conflicting contingencies.

1.2.4 Distributed and Centralized Planning and Scheduling

Another important issue dealt with in this report is the distinction between **centralized** and **distributed** planning and scheduling.

- A centralized resolution of planning and scheduling problems consists of making all the decisions in one place. All the orders are collected by a central task planner and scheduler (CTPS). The CTPS is a computer dedicated to the resolution of planning and scheduling problems. It creates a plan for each order, decides which resources to use, orders actions when necessary, and assigns them to agents with respect to their work load, capabilities and location.

- In a distributed framework, the decision-making of a group of agents is a product of the expectations and expertise of the individual agents, "resolved through a process of conflict resolution, cooperation, bargaining, or negotiation" [1]. In the planning and scheduling domain, an extreme possibility is to make agents accept orders from clients and exchange tasks or sub-tasks by one-to-one negotiation. No CTPS is needed. A particular initial decomposition of the global planning and scheduling problem is imposed and agents negotiate one with the other to make the best of potential interactions and solve conflicts.

A centralized architecture is theoretically attractive. It enables the anticipation of interactions and allows any decomposition of the overall planning and scheduling problem. The ability to choose a decomposition with respect to the characteristics of the problem at hand is significant. Indeed, there is no best pre-determined decomposition of the overall planning and scheduling problem. Results of experiments in the domain of job-shop scheduling [106] provide evidence that considering multiple problem decompositions allows to enhance both the efficiency of the scheduling process and the quality of executed schedules. On the other hand, a centralized approach is (for example) not appropriate for coordinating the actions of mobile agents in dynamic environments where unforeseeable events occur. Mobile agents are not always in contact with the CTPS, while the CTPS needs information about the actual course of events to be effectively reactive. Solutions of this problem (e.g. placing sensors all over a building) are difficult and costly enough to implement for us to consider distributed approaches.

Unfortunately, scheduling problems are tightly coupled. They can be decomposed into subproblems but a lot of interactions exist between subproblems. A completely distributed architecture makes the anticipation of these interactions difficult and entails the emergence of many conflicts in real-time.

This leads to the examination of intermediate architectures. An appealing approach is to distribute planning and scheduling decisions, but to ask agents to refer to a common "deductive database" [50]. Each agent is a problem-solving component which chooses the tasks it wants to perform and explains how it will perform them (provided that time is available for this explanation). The deductive database is a solution maintenance component which fulfills two objectives.

- It provides a characterization of the set of alternatives that exist with respect to planning and scheduling decisions that remain to be made. This allows an agent to examine planning and scheduling alternatives and determine which are the most appropriate given the plans of other agents.
- It detects conflicts between goals, decisions, preferences, reported events and predicted events. This allows to warn an agent of previously unforeseen interactions when communication with the agent is possible.

Another solution is to centralize predictive activities and distribute reactive activities. In this case, an agent which reacts to a perturbation needs to determine which actions can be executed without disrupting the global schedule. In a similar fashion, a CTPS can make global decisions (e.g. choose which orders to execute first) while individual agents handle the details in a distributed fashion. Such a decomposition of the overall planning and scheduling effort is particularly appropriate when the most frequent and significant interactions are detectable (and avoidable) at a high level of abstraction.

2 Theoretical Foundations for Constraint Propagation Systems

Following [61], we distinguish three lines of research in constraint-based reasoning. Constraint languages can be emphasized as (1) adequate knowledge representation formalisms, (2) means to solve combinatorial problems and (3) convenient computational models. These three views suggest different criteria to compare constraint-based systems:

- A good knowledge representation formalism allows a simple and precise representation of constraints for the widest collection of problems. Describing problems in a constraint language allows to determine how good it is as a knowledge representation formalism.

- The second view suggests that a constraint system is all the more useful as it speeds up problem-solving and enables the generation of better problem solutions for the widest collection of problems. Both theoretical and experimental comparisons are of interest. However, one must consider the results with caution. High (e.g. $O(n^3)$) worse-case or average-case algorithmic complexities do not mean that the corresponding algorithms are slow when applied to particular classes of real problems (see section 5). Similarly, experimental results showing that a simple heuristic method is the most efficient to solve small problems do not prove that its performance remains acceptable as the problem size increases (see [75]).

- The previous criteria allow to compare computational models. A computational model is convenient for a collection of problems when it enables a simple description and an efficient resolution of the considered problems. However, one expects more from a constraint language considered as an implementation of a computational model: (a) a precise assessment of the language semantics and (b) a theoretical completeness result. In this respect, it is interesting to compare the two most well-known constraint logic programming frameworks. The CLP framework [69] requires that the constraint-solver itself is complete while in CHIP [149] completeness is achieved through search (and the programmer is allowed to specify how to propagate a constraint). The behavior of a CLP program is easier to understand and anticipate, but the CLP completeness requirement prevents the realization of effective applications in the planning and scheduling domain.

In this report, we compare various constraint propagation methods considered as tools to solve combinatorial problems. In most cases, the methods that we compare are equivalent with respect to the two other views: the same syntax is adopted for the same constraints (with the same semantics) and we never suppose that the constraint propagation system is complete. Constraint propagation is in all cases defined as a deductive activity performed by a "solution maintenance component" for one or several "problem-solving components". Problem-solving components may be called modules, subsystems, knowledge sources or agents, depending on the overall system architecture. With respect to [132] and [29], we refer to them as "system components" producing "solution components" to be integrated.

The solution maintenance component derives new constraints from existing ones and detects conflicts (inconsistencies) between constraints. It provides a characterization of choices that remain to be made and a description of detected inconsistencies. Problem-solving components make and retract decisions accordingly.

In this section, we (a) present the theoretical foundations underlying the constraint propagation methods discussed in this report and (b) provide an overview of the constraint language. First, we discuss the most usual definitions of "validity", "completeness" and "decidability" of inference systems and show that these definitions are inadequate for solution maintenance components (section 2.1). This leads us to propose new definitions which make sense because they refer to the needs of problem-solving components (section 2.2). Then we provide examples of complete theories for solution maintenance components (section 2.3) and shortly discuss the utilization of these theories (section 2.4). [87] provides more details about the overall constraint propagation system. [31] and [38] present its principal applications.

2.1 Usual Definitions of Validity / Completeness / Decidability

Constraint propagation is a deductive activity performed by a solution maintenance component for problem-solving components. To discuss the validity, completeness and decidability properties of solution maintenance components, we first refer to the most usual definitions of validity, completeness and decidability [98]:

- An inference system I is **valid** if whenever there is a derivation of a sentence s from a set of sentences S_0 by means of inference rules and/or axioms in I, s is true in every model of $(I \cup S_0)$.
- An inference system I is **deduction-complete** for a set of sentences S (usually the set of all sentences constructible in the language of I) if whenever a set of sentences S_0 logically implies a sentence $s \in S$ (i.e. whenever s is true in every model of $(I \cup S_0)$), there is a derivation of s from S_0 by means of inference rules and/or axioms in I.
- An inference system I is **refutation-complete** if whenever a set of sentences S_0 logically implies a contradiction (i.e. whenever there is no model of $(I \cup S_0)$), there is a derivation of an effectively recognizable contradiction from S_0 by means of inference rules and/or axioms in I.
- An inference system I is **deduction-decidable** for a set of sentences S if there exists a finite proof procedure which, whenever a set of sentences S_0 logically implies a sentence $s \in S$, eventually generates a derivation of s from S_0 by means of inference rules and/or axioms in I.

- An inference system I is **refutation-decidable** if there exists a finite proof procedure which, whenever a set of sentences S_0 logically implies a contradiction, eventually generates a recognizable contradiction from S_0 by means of inference rules and/or axioms in I.

But the goal of constraint propagation is not to prove sentences from a set of initial sentences, but to determine constraints the satisfaction of which is necessary to satisfy a set of initial constraints incrementally provided by a problem-solver. When a solution maintenance component deduces $(x \leq 3)$ from a set of initial constraints C, it indicates that the value of x must be less or equal to 3 to enable the satisfaction of constraints in C. If this information is not exploited by some problem-solving component (or by the solution maintenance component itself), it should not be deduced. A new intuitive definition of deduction-completeness follows: a solution maintenance component is deduction-complete with respect to a problem-solving component if and only if it provides all the information the problem-solving component may use. Two problems arise:

- A problem-solving component able to use $(x \leq 3)$ does not need the solution maintenance component to deduce $(x < 4)$, $(x < 5)$, $(x < 6)$ and so on from $(x \leq 3)$. The new constraint $(x \leq 3)$ implicitly contains all the constraints $(x < n)$ for $(n > 3)$. If a formal definition of "implicitly contains" can be provided (i.e. if the solution maintenance component and the problem-solving components can agree on "what is implicitly contained in what"), then a new formal definition of deduction-completeness, which corresponds to the needs of problem-solving components, can be used.

- We must discuss the validity and the completeness of a solution maintenance component with respect to the set of interpretations in which problem-solving components are interested. If a problem-solving component deals with inequalities between variables and integers and assumes variables will eventually "pick" integers, the solution maintenance component can and must use a rewriting rule as "$(x < y) \Rightarrow (x + 1 \leq y)$" (or equivalent rules) to be complete. But such a rule will not be valid with respect to another problem-solving component allowing a continuous range of values for its variables. Similar problems are often encountered during the design and the implementation of expert systems [116]: the set of rules which constitute the knowledge base of an expert system is valid and complete if and only if the expert and the knowledge base allow the same interpretations. In some sense, the underlying problem is simpler in the case of constraint propagation since a formal definition of allowed interpretations can often be provided to enable a new definition of validity and completeness.

2.2 Revised Definitions (for Solution Maintenance Components)

Consequently, we define a theory of constraint propagation as a tuple $(P\ R\ D)$. P is a set of **constraint construction rules** defining well-formed constraints. For example, "if n is an integer and if e_1 ... e_n are well-formed constraints, then $(or\ e_1\ ...\ e_n)$ is a well-formed constraint" is a construction rule. R is a set of **interpretation rules** defining the set of allowed interpretations. For example, "(*or* e_1 ... e_n) is satisfied if and only if e_i is satisfied for some i in $\{1 \ldots n\}$" is an interpretation rule. D is a set of **deduction rules** (or axioms) explicitly associated with propagation activities and constraint construction rules. Three principal constraint propagation activities are distinguished:

- **Combination** consists in building a new constraint from a set of existing constraints. For example, the generalized resolution rule "if g can be derived from e_i and f, then (*or* e_1 ... e_{i-1} g e_{i+1} ... e_n) can be derived from (*or* e_1 ... e_i ... e_n) and f" is associated with the constraint combination activity and the *or* constraint construction rule.

- **Subsumption** allows to hide a constraint the satisfaction of which results from the satisfaction of another constraint. For example, the subsumption rule "if e subsumes e_i for some i in $\{1 \ldots n\}$, then e subsumes (*or* e_1 ... e_n)" is associated with the subsumption activity and the *or* constraint construction rule. A particular case of subsumption consists in hiding tautologies as rules conclude "T subsumes c".[9]

- **Rewriting** allows to write constraints in normal forms and to translate constraints from a representation to another. For example, "$(not\ (x \leq_E y)) \Rightarrow (y <_E x)$" is associated with the rewriting activity and the constraint construction rules *not*, $\leq_E$ and $<_E$ as soon as $(E \leq)$ is defined as a totally ordered set.

Given a set of constraints C, a model of C in $(P\ R\ D)$ is an interpretation which satisfies every constraint in C and every interpretation rule in R. $M(C)$ denotes the set of all the models of C and $D(C)$ denotes the set of constraints derivable from C by means of propagation rules and/or axioms in D. New definitions follow:

- $(P\ R\ D)$ is **valid** if $\forall C$, $M(D(C)) = M(C)$.

- $(P\ R\ D)$ is **subsumption-valid** if whenever $\bar{c}$ subsumes c, $M(c)$ contains $M(\bar{c})$.

- $(P\ R\ D)$ is **refutation-complete** if whenever $M(C)$ is empty, there is a derivation of NIL (an "effectively recognizable contradiction") from C by means of inference rules and/or axioms in D.

- $(P\ R\ D)$ is **refutation-decidable** if there exists a finite propagation procedure which, whenever $M(C)$ is empty, generates a derivation of NIL from C by means of inference rules and/or axioms in D.

- A constraint c **implicitly belongs** to a set of constraints C if $C \cup \{T\}$ contains a constraint $\bar{c}$ which subsumes c.

[9]Kelly, Steinberg and Weinrich discuss constraint subsumption (and the need to perform tautology tests) in the context of VEXED, a design aid for NMOS digital circuits [140] [141] [72]: "It is not clear whether the existing subsumption test code actually speeds up EVEXED at all". In job-shop scheduling, the subsumption of inequalities is necessary. On the other hand, experimental results concerning the subsumption of disjunctive constraints (see section 5) do not allow us to draw very specific conclusions.

- $(P\ R\ D)$ is **deduction-complete** for a set of constraints S (usually the set of all constructible constraints) if whenever a constraint $c \in S$ is satisfied in every interpretation in $M(C)$, c implicitly belongs to $D(C)$.
- $(P\ R\ D)$ is **deduction-decidable** for a set of constraints S if there exists a finite propagation procedure which, whenever a constraint $c \in S$ is satisfied in every interpretation in $M(C)$, eventually generates a derivation of a constraint $\bar{c}$ (maybe T) which subsumes c.

Compared to standard definitions, these definitions refer to the implicit belonging of a constraint c to a set C and to interpretation rules. However, the formal definition of "implicitly belongs" does not seem, at the first glance, to match the intuitive definition proposed in section 2.1. The intuitive definition refers to the capabilities of the current user of the solution maintenance component, i.e. to the capabilities of a problem-solver. $(x \leq 3)$ implicitly contains $(x < 4)$ if the problem-solver does not need $(x < 4)$ when it knows $(x \leq 3)$. For an idiotic problem-solver, we are reduced to the explicit belonging of c to C and to a more usual definition of deduction-completeness and decidability.

The formal definition corresponds to the use of problem-solvers in which the contents of subsumption rules are somehow "integrated". This makes sense because the solution maintenance component may be allowed to hide $(x < 4)$ in the presence of $(x \leq 3)$ only if the problem-solver does not need $(x < 4)$ in the presence of $(x \leq 3)$. In other terms, if c implicitly belongs to C with regards to the formal definition, then c implicitly belongs to C with regards to the intuitive definition. On the other hand, if "c implicitly belongs to C with regards to the intuitive definition" does not imply "c implicitly belongs to C with regards to the formal definition", new subsumption rules may be written to fill the gap between the two definitions.

2.3 Examples of Complete Theories

The implementation of a theory equivalent to first-order logic (refutation-complete but not decidable as a consequence of Church's theorem [77]) is possible, but we did not implement such a theory. The expressive power of first-order logic is generally not needed as far as constraint propagation is concerned. Introducing convenient sets of propagation rules ("operators" [128]) through more specific axiomatizations and using them when necessary is much more efficient. In our system [87], many complete theories of constraint propagation are available:

- A refutation-complete and decidable theory for propositional logic. The theory is also deduction-complete and decidable for atomic formulas and for formulas built from atomic formulas by the means of the *not* constraint construction rule as (*not* (*not* P)).

- A refutation-complete and decidable theory for point ordering relations in infinite totally ordered sets [150].[10] The theory is easily extended to allow disjunctive, conjunctive and negative constraints as $(or\ (x \leq y)\ (not\ (z \leq y)))$ and stay refutation-complete and decidable. Furthermore, a problem-solving component can commit to a particular totally ordered set without endpoints $(E \leq)$ and manipulate constants from this set as soon as it provides a definition of the ordering relation and indicates which elements of E are immediate predecessors or successors of other elements of E.

- A refutation-complete and decidable theory for James Allen's interval relations [4] and disjunctions of interval relations. The theory is easily extended to allow any disjunctive, conjunctive and negative formula of interval relations and stay refutation-complete and decidable.

- A refutation and deduction-complete and decidable theory for duration constraints (minimal, maximal and known distances between time points). Our only assumption about durations is that they belong to the positive part of a totally ordered Abelian group (e.g. the set of integers Z or $Z * \{0 \ldots 23\} * \{0 \ldots 59\}$) defined by the user of the system. The theory is easily extended to allow any disjunctive, conjunctive and negative formula of duration constraints and stay refutation-complete and decidable.

- A refutation and deduction-complete and decidable theory for reservation constraints. Basically, each reservation constraint $(reserve\ G\ t_1\ t_2\ n\ motives)$ states that n individual resources from the group G are unavailable throughout the interval of time $(t_1\ t_2)$. The list of motives explains why they are unavailable: operations or maintenance periods are scheduled during the interval $(t_1\ t_2)$, resources are down, or there is no work shift planned over the interval $(t_1\ t_2)$.[11] Constraint propagation enables the maintenance of accurate time-tables and the detection of capacity conflicts (e.g. over-

[10]This theory (based on [150]) **is not deduction-complete.** In [150], Vilain and Kautz erroneously state the (deduction-)completeness of the proposed algorithm. Le Pape [87] reproduces the error. Van Beek presents a counter-example in [146] and proposes real deduction-complete algorithms in [146] and [147]. Ghallab and Mounir Alaoui use "complete" for "refutation-complete" in [55] and discuss counter-examples in [56]. Their approach requires that the network of relations is kept consistent. Refutation-completeness is needed to detect a conflict and discard a constraint as soon as it is inconsistent with the others. But deduction-completeness is not needed for this purpose.

[11]Here t_1 and t_2 are constants. This means that given two (or more) activities requiring resources from the same group, one must assign start and end times to these activities in order to use reservation constraints. When a group consists of a unique resource, an alternative technique consists in generating a disjunctive constraint $(or\ ((end\ A) \leq (start\ B))\ ((end\ B) \leq (start\ A)))$ for each pair of activities $\{A\ B\}$ that require the resource. This allows the start and end times to fluctuate. In practice, the generalization of this technique is not possible: the size of each disjunction grows to $(n+1)!$ for a group of n resources. [87] proposes a simple but incomplete set of propagation rules to deal with reservations and inequalities at the same time. In a similar spirit, [43] proposes the notion of energy consumption to extend the constraint propagation techniques presented in [42] and [44]. The basic idea is simple: the energetic consumption over a time interval cannot exceed the amount that is available over this interval. This allows the deduction of temporal inequalities from energetic considerations.

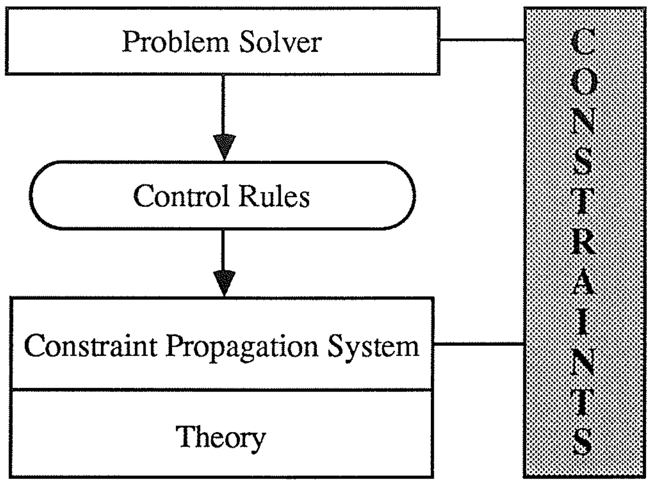

Figure 6: Controlling Constraint Propagation

loads). The theory is easily extended to allow any disjunctive and conjunctive formula of reservation constraints and stay refutation-complete and decidable.

- A refutation-complete and decidable theory (not discussed in [87]) for several families of domain constraints as $(x \in \{a_1 \ ... \ a_n\})$, together with equalities as $(x = y)$ and $(x = 2)$, and inequalities as $(x \leq y)$, $(x < y)$ and $(2 < x)$ as soon as the overall domain (defined by the user of the system) is totally ordered. Disequality constraints as $(x \neq y)$ and $(x \neq 2)$ can also be confronted to domain constraints. However, the general constraint consistency problem is NP-hard when disequality constraints are allowed in finite domains. In such cases, the best way to proceed (if a refutation-complete constraint propagation procedure is definitely desired) is either to rewrite domain constraints $(x \in \{a_1 \ ... \ a_n\})$ into disjunctions of equalities $(\mathit{or} \ (x = a_1) \ ... \ (x = a_n))$, or to rewrite disequality constraints $(x \neq y)$ into disjunctions of inequalities $(\mathit{or} \ (x < y) \ (y < x))$ if the overall domain is totally ordered. The resulting theory extends a set of propagation rules discussed in [148]. We are currently interested in various families of 2D domain constraints for which no efficient and complete system is known yet.[12]

For each of the developed theories, a refutation-complete constraint propagation procedure is available (often deducible from constructive demonstrations in [87]). The overall system described in [87] is refutation-complete and decidable.

2.4 Conclusion

Section 2.2 provides new definitions for the validity, completeness and decidability of solution maintenance components. Given a problem (or a problem-solver), these definitions, together with validity, completeness and decidability results, facilitate the selection of a set

[12]Rit's constrained occurrences problem [113] is a good example. Indeed, rewriting interval relations and generalized window domain constraints into conjunctions and disjunctions of duration constraints enables the achievement of refutation-completeness, but significantly affects efficiency in simple cases.

of propagation rules which determine what inferences may be drawn. The next step in adapting constraint propagation to a particular problem is discussed in [24]. It consists in writing control rules which collectively determine what inferences must be drawn and in what order they must be drawn (figure 6). Control rules define a proof procedure called "constraint propagation scenario". Of course, the completeness of a theory $(P\ R\ D)$ does not force us to use a complete constraint propagation scenario. As in logic and theorem-proving, it means any logically implied constraint and any contradiction can be recognized as such if we actually want to. But there is a trade-off with respect to the amount of computational effort spent in constraint propagation. The general problem of determining whether some planning and scheduling decisions are compatible is NP-hard. This highly suggests the adoption of an incomplete constraint propagation scenario.

3 Type and Domain Constraints in Task Planning

Generally speaking, "situation-independent" and "situation-dependent" task planning steps can be distinguished. Given a goal, we can construct plan structures with no information about the current situation of agents and objects in the considered environment. If the goal is to provide John with a cup of coffee, a plan structure specifies that some agent could go to some vending machine, get a cup of coffee from the vending machine and bring it to John. On the other hand, deciding which particular agent will execute which actions with which resources requires an assessment of the specific situation at hand.

This section discusses a task planning system allowing the performance of situation-independent planning steps. The advantage of such a system is that plan structures for a goal are derived once and for all using no situation-specific information. Agents willing to contribute to the achievement of a goal instantiate the plan structures with respect to their knowledge of the current situation. If execution fails, there is no need to re-derive plan structures. Depending on the reason for the failure, agents can execute the same instance again or instantiate the plan structure with respect to the new situation.

An attractive idea to generate and represent situation-independent plan structures is to use type specifications (e.g. $(type\ ?R) = (or\ robot\ workstation)$) in place of domain constraints (e.g. $?R \in \{robot_1\ ...\ robot_m\ workstation_1\ ...\ workstation_n\}$). However, domain constraints are easier to propagate than type specifications. In practice, this means there is a trade-off between the generality of the constructed plan structures and the speed of the planning process. On one extreme, all the theoretical possibilities are accounted for and plan structures never need to be revised. On the other extreme, plan structures are built with respect to situations from which execution could practically start.

In this section, we describe experiments allowing to assess (for a particular domain) the computational behavior of a few variations of the same task planning system, able to propagate both type specifications and domain constraints. Section 3.1 explains in which context this work has been performed and presents the task planning system. Section 3.2 discusses constraint propagation and defines constraint propagation scenarios to compare in this context. Section 3.3 presents the considered task planning problems and section 3.4 discusses the experimental results.

3.1 A Task Planner Based on Constraint Propagation

The work presented here and in section 4 has been performed in the context of two projects aimed at controlling the actions of multiple interacting agents:

- The goal of the first project is to control the operations of many mobile robots (several dozens) in an indoor environment (an office environment, a shop-floor, an airport, an hotel) in order to automate a variety of tasks. Typical tasks include transportation of objects (beverages, books, mail), operation of machines (copiers, vending machines), cleaning and maintenance. Most of the tasks essentially involve mobility and transportation of relatively small objects. Nevertheless, some classes of tasks may require other physical capabilities. For example, tasks related to the operation of a photocopier may be performed with the help of a fixed manipulator robot standing near the copier. In addition to operating the copier (loading documents to be copied, selecting the appropriate options, adding toner when needed, etc), the fixed robot may also load and unload mobile robots transporting the documents.

- The second project concerns the integration of various short-term planning and execution monitoring techniques for multiple agents (robots and humans) working on a construction site. The case of a construction site introduces additional difficulties. For example, the geometry of a construction site continually changes. More sophisticated planning techniques (integrating temporal and geometrical reasoning) are consequently needed.

The task planning problems considered in this section concern mostly the first — more advanced — project. The current planning and execution system integrates a collection of software components: a **task planning system** to derive plans made of "high-level" actions (such as "go to position P" or "get object O") from a description of the tasks to be performed; a **task allocation system** to order and allocate tasks and actions to robots (see [90] and section 4); a **motion planning system** to convert "high-level" actions into motion commands [15]; and an **execution system** to monitor execution and react to unexpected events [16]. These components are implemented in COMMON-LISP on a DEC 3100 workstation. The overall system is tested with the help of a simulator specially designed to simulate actions of autonomous agents [91].

Task planning consists of determining programs of actions that can be carried out to achieve goals. An agent can acquire goals in two ways. The first is to offer or to be asked to achieve goals of other agents. The second is to autonomously generate goals with respect to the current situation. For example, idle mobile robots should contact other agents through the communication network to determine what they could do for them. A mobile robot generates a goal ((*on-line myself*) = *T*) as soon as it is free of other goals.

We distinguish plan structures, which contain variables, and plan instances, which are instantiated plan structures. A plan structure is defined as an action hierarchy and a set of constraints.

- **Action Hierarchy**. The top-level action represents the entire process to carry out. Each action can be refined into either a sequence of more detailed actions, a set of un-sequenced actions (to be performed either in parallel or in any order), or a set of exclusive alternatives. The actions at the bottom of the hierarchy are atomic formulas containing variables. For example, (*connect* $?R$ $?P$) corresponds to the connection of any robot $?R$ with any communication network port $?P$.

- **Constraints** provide a specification of admissible assignments of values to variables. Type specifications (e.g. (*type* $?R$) = (*or robot workstation*)) or domain constraints (e.g. $?R \in \{robot_1 \; ... \; robot_m \; workstation_1 \; ... \; workstation_n\}$) describe domains over which variables can vary. Variable relations (e.g. (*location* $?R$) = $?L$) define a subset of the cartesian product of these domains.

A plan instance is an action hierarchy in which variables have been replaced by values satisfying the constraints. For example, [((*type* $?P$) = *port*) (*sequence* (*move myself* $?P$) (*connect myself* $?P$))] is a plan structure for the goal ((*on-line myself*) = *T*). If $port_{22}$ is a particular port, then (*sequence* (*move myself* $port_{22}$) (*connect myself* $port_{22}$)) is an instance of this plan structure.

Given a new goal, we consider three planning steps. In the first step, the robot determines plan structures to achieve the goal. Plan structures are either retrieved from a library (as in [53] and [54]) or constructed with the help of the task planning system. The plan construction process consists in rewriting a formula describing the plan under development. In the connection example, the original formula is set to (*achieve* ((*on-line myself*) = *T*)). There are two ways to rewrite an *achieve* formula. The first is to post a constraint stating that the goal statement (i.e. (*on-line myself*) = *T*) is true in the initial situation. The *achieve* formula is then reduced to the empty string. The second is to assume the goal statement is false in the initial situation (i.e. (*on-line myself*) $\neq$ *T*) and use an **operator** to make it true. Each operator corresponds to an executable action and is defined as a statement containing variables, together with a set of constraints, preconditions, and effects. For each operator and each effect, the unification of the effect and the goal statement determines constraints under which the operator allows to achieve the statement. These constraints are posted and (if no contradiction is detected) the *achieve* formula is rewritten as a sequence (*sequence* (*achieve preconditions*) *operator-statement*). Two **contexts** referring to the state of the world before and after the action are associated with the operator statement. The preconditions and the constraints from the operator are posted in the context preceding the action and the effects are posted in the context following the action. The variables of the operator are renamed to avoid confusions (in case the same operator appears twice in the same plan structure) and the rewriting process continues unless the set of posted constraints is determined inconsistent. In the connection example, the *connect* operator defined as follows provides a possible unification.

- Statement: (*connect* $?R$ $?P$)
- Constraints: {((*type* $?P$) = *port*) ((*type* $?R$) = *robot*)}
- Preconditions: {((*location* $?R$) = $?P$)}
- Effects: {((*on-line* $?R$) = *T*)}

The formula becomes (*sequence* (*achieve* ((*location myself*) = $?P_1$)) (*connect myself* $?P_1$)). Five constraints are posted and two of them disappear: ($?R$ = *myself*) results in the replacement of the variable $?R$ with the constant *myself* everywhere in the formula and in the constraint base; then ((*type myself*) = *robot*) is evaluated (rewritten) into the empty constraint T. The distinction between the constraint set and the preconditions determines which statements the task planner tries to achieve with other operators (preconditions). There is a unique precondition in the *connect* operator. In most cases, there are several preconditions related with three possible connectives: *sequence*, *alternative* and *parallel* (the default case).

- In the first case, the task planner tries to achieve the first precondition from the initial situation, then the second precondition from the resulting situation, and so on. In other terms, (*achieve* (*sequence* p_1 ... p_n)) is rewritten as (*sequence* (*achieve* p_1) ... (*achieve* p_n)) and the different *achieve* formulas are considered in chronological order.

- In the second case, the task planner needs to achieve one precondition out of the disjunction. It determines whether one precondition is true in the initial situation. Otherwise it considers the addition of a new constraint and the use of operators for each alternative.

- In the third case, the task planner considers the different preconditions in all possible orders. This means it assumes that solving one precondition (out of the conjunction) and then following this solution with the solution to the other preconditions will be successful. This assumption (called "linearity assumption" [64]) is valid in the planning domains we consider. But it is not valid in other domains such as the famous "blocks world" [105].

In several cases, there are different possibilities to rewrite an *achieve* formula: post a constraint or use an operator (several operators could work); try alternative preconditions; order parallel preconditions. A depth-first search algorithm is used to explore the different possibilities. It stops when all the possible plan structures have been determined (it could also stop as soon as one possible plan structure has been found). Finally, the task planning system allows to determine the unsuitable effects of a plan and post goals to undo these effects. For example, when a mobile robot borrows a book (to make a copy of it), a goal to return the book to its original location is generated and treated as a post-condition to achieve.

In the second step, the robot instantiates plan structures. It uses a constraint satisfaction algorithm which allows to determine *one*, *all*, *the best* or *the n best* substitutions of variables, with respect to some user-provided evaluation function. For example, it allows to determine the substitution which minimizes an estimate of the total duration of the plan. The algorithm is a simple depth-first search algorithm. Given a plan structure, it selects a variable $?V$ to instantiate and a possible value *value* at each step. It posts the constraint ($?V$ = *value*) and waits for the constraint propagation algorithm to determine the consequences of this instantiation. Then it proceeds with the remaining variables. Domain-dependent heuristics can be used to select variables and values and to prune the search space. In the absence of such heuristics, the system (a) avoids the selection of a variable whose value is derivable from the value of another variable (for example, if ((*location* ?R) = ?P) is a constraint, the value of ?P is derivable from the value of ?R), (b) selects among the remaining variables the variable with the smallest domain and (c) selects a random value from this domain.

In the third step, the robot makes final decisions. Depending on its planning policies, the robot may have developed several goals, several plan structures for the same goal and several plan instances for the same plan structure. The general problem is very complex since plans developed to achieve individual goals are likely to interact. We ignore this problem here and concentrate on the first two steps. In practice, robots can use a simple user-provided evaluation function to choose one plan to execute prior to the others.

3.2 Constraint Propagation

Various types of constraints are posted during the task planning process: type or domain constraints; equalities and disequalities, either between constants and variables, or in an "object-attribute-value" form ((*attribute object*) = *value*); and any other type of constraint appearing in operator descriptions. In this section, we consider constraint propagation in the simplest case in which constraints other than type constraints, domain constraints, equalities and disequalities are not worth propagating. For each constraint type, we discuss possible constraint propagation steps (section 3.2.1). Then we determine constraint propagation scenarios to compare (section 3.2.2).

3.2.1 Constraint Propagation Steps

Object-Attribute-Value Relations

An important problem to consider is that "object-attribute-value" relations refer to dynamic properties of the considered objects. For example, the action (*connect myself* ?P) changes the value of (*on-line myself*). Constraints in the "object-attribute-value" form are posted with respect to some context: they refer to the state of the world either before or after the execution of an action. We use the syntax (*in-context* C ((*attribute object*) = *value*)) to precise in which context the relation is required to hold. The context I refers to the initial context at the beginning of the plan and the context F refers to the final context at the end of the plan. The following rule allows to combine relations holding in the same context:

- The disjunction (*or* ($object_1 \neq object_2$) ($value_1 = value_2$)) follows from (*in-context* C ((*attribute* $object_1$) = $value_1$)) and (*in-context* C ((*attribute* $object_2$) = $value_2$)).[13]

[13]Let us note that ((*attribute object*) $\neq$ *value*) is equivalent to the conjunction of two constraints ((*attribute object*) = $\overline{value}$) and (*value* $\neq$ $\overline{value}$). We do not need to consider "object-attribute-value" disequalities.

In addition, the system propagates a relation from a context to another as soon as a context C preceding an action is identified with another context, either I (when the action is the first action of the plan), or another context D ending the preceding action.

Let us first consider the case of the initial context. The planning agent usually has some partial knowledge about the possible initial situations. Given the constraint (*in-context* I ((*attribute object*) = *value*)), we can consider various constraint propagation steps:

- The previous rule can be applied with (*in-context* I ((*attribute object*) = *value*)) and any piece of knowledge ((*attribute* $object_i$) = $value_i$) about the initial situation. This is the easiest propagation rule to implement. However, we can expect its use to be very costly since it generates a lot of disjunctive constraints to satisfy.

- An alternative is to restrict constraint propagation to cases in which either *object* or *value* is a constant. If *object* is a constant, (*in-context* I ((*attribute object*) = *value*)) and ((*attribute object*) = $value_i$) provide (*value* = $value_i$). If *object* is a variable and *value* a constant, we can determine whether there are instances $object_i$ in the domain of *object* such that the initial state does not contradict ((*attribute* $object_i$) = *value*). We can also eliminate from the domain the instances $object_i$ such that (*attribute* $object_i$) differs from *value*.[14]

- Another possibility is to (a) make the propagation steps mentioned above when either *object* or *value* is a constant and (b) achieve "arc-consistency" when both *object* and *value* are variables to which domain constraints are associated. "Arc-consistency" is achieved when the domains of *object* and *value* are reduced to the maximal domains *dom*(*object*) and *dom*(*value*) included in the original domains and such that (1) for each instance $object_i$ in *dom*(*object*), there is an instance $value_i$ in *dom*(*value*) such that ((*attribute* $object_i$) = $value_i$) is possible and (2) for each instance $value_i$ in *dom*(*value*), there is an instance $object_i$ in *dom*(*object*) such that ((*attribute* $object_i$) = $value_i$) is possible.

The case in which a context C preceding an action A is identified with a context D ending the preceding action B is much simpler. Let (*in-context* C ((*attribute object*) = *value*)) be the relation of interest to us. For each effect of B referring to the same attribute *attribute*, (*in-context* D ((*attribute* $object_i$) = $value_i$)), with ($1 \leq i \leq n$), the disjunctive constraint (*or* (*object* $\neq$ $object_i$) (*value* = $value_i$)) is generated. In addition, the disjunctive constraint (*or* (*object* = $object_1$) ... (*object* = $object_n$) (*in-context* E ((*attribute object*) = *value*))), where E denotes the context preceding B, is also generated to express the fact that a relation

[14]The planning agent does not remove $object_i$ from the domain when (*attribute* $object_i$) is unknown. It could be the case that the unknown — but existing — value of the attribute meets the constraint. Another particular situation is when the attribute *attribute* is not applicable to $object_i$: the value does not exist. Then $object_i$ cannot meet the constraint. It is discarded from the domain. The reader interested in the distinction between non-applicable nulls and unknown values — and their manipulation — in relational databases and knowledge-based systems can refer to [32] [93] [94].

untouched by an operator application — and true after the operator application — was true before the operator application. The generation of all these constraints is necessary for the planner to correctly assess the effects of each operator application.

Equalities and Disequalities

Equalities and disequalities are the easiest constraints to propagate. Four cases can occur for an equality:

- An equality $(x = x)$, where x is a variable, or $(a = a)$, where a is a constant, is a tautology (rewritten into T). It is not propagated any further.
- An equality $(x = y)$, where x and y are two distinct variables, results in the replacement of one variable (e.g. y) with the other variable everywhere in the constraint base. Propagation continues with the modified constraints.
- An equality $(x = a)$ or $(a = x)$, where x is a variable and a a constant, results in the replacement of x with a everywhere in the constraint base. Propagation continues with the modified constraints.
- An equality $(a = b)$, where a and b are two distinct constants, is a contradiction (rewritten into NIL). Propagation stops and the contradiction is returned to the task planner.

Similarly, four cases can occur with disequalities:

- A disequality $(x \neq x)$, where x is a variable, or $(a \neq a)$, where a is a constant, is a contradiction (rewritten into NIL). Propagation stops and the contradiction is returned to the task planner.
- A disequality $(x \neq y)$, where x and y are two distinct variables, can combine with or subsume disjunctions: when a disjunction contains a disjunct $(x \neq y)$ or $(y \neq x)$, the disequality subsumes the disjunction; when a disjunction contains a disjunct $(x = y)$ or $(y = x)$, a simpler disjunction (obtained by removing the disjunct) is obtained.
- A disequality $(x \neq a)$ or $(a \neq x)$, where x is a variable and a a constant, can combine with or subsume disjunctions (exactly as above). If a domain constraint is associated with x, we can also combine the disequality and the domain constraint, i.e. eliminate a from the domain of x.
- A disequality $(a \neq b)$, where a and b are two distinct constants, is a tautology (rewritten into T). It is not propagated any further.

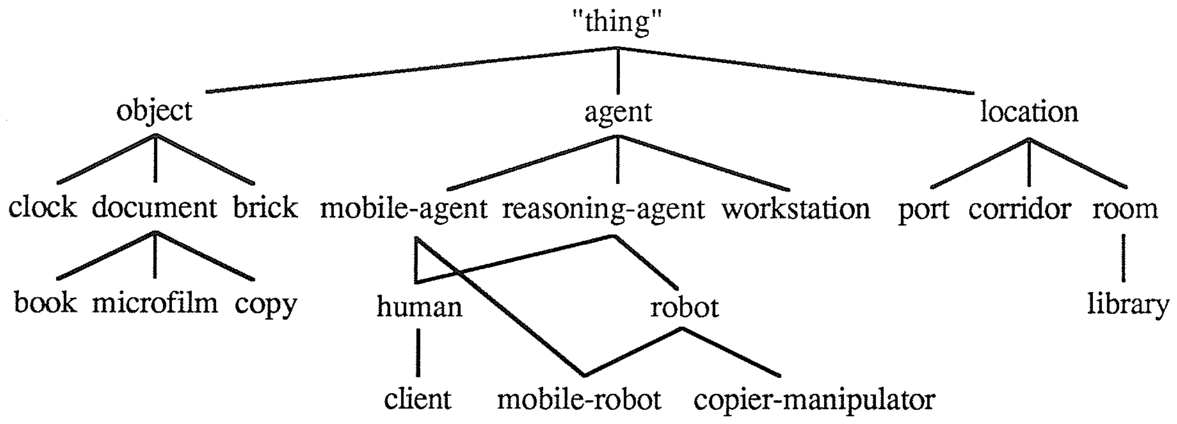

Figure 7: A Hierarchy of Types

Type and Domain Constraints

When two type or domain constraints concern the same variable x (e.g. this happens when an equality constraint between two variables results in the replacement of one variable), their combination provides a new — more precise — type or domain constraint which subsumes the two original constraints.

In the case of type constraints, we suppose that the constraint propagation system has been provided with a hierarchy of types (e.g. figure 7). A type specification is a disjunction of pairwise incomparable types (e.g. (*or robot workstation*)).[15] As explained in [2] [3] [118], the type hierarchy induces a partial order $\subseteq$ on types ($t_a \subseteq t_b$ if and only if t_a is a subtype of t_b) which can be extended to a partial order $\sqsubseteq$ on type specifications: $T_A \sqsubseteq T_B$ if and only if for each disjunct t_a in T_A, there is a disjunct t_b in T_B such that t_a is a subtype of t_b. The set of type specifications provided with this partial order is a lower semilattice and the combination of two type constraints $(type\ x) = T_A$ and $(type\ x) = T_B$ provides a new type constraint $(type\ x) = T_A \sqcap T_B$ where $T_A \sqcap T_B$ denotes the greatest lower bound of T_A and T_B in the semilattice. The new constraint subsumes both $(type\ x) = T_A$ and $(type\ x) = T_B$. It is rewritten into NIL when $T_A \sqcap T_B$ equals $\emptyset$.

The case of domain constraints is much simpler since the intersection of the domains can be directly computed. The combination of $(x \in A)$ and $(x \in B)$ provides a new constraint $(x \in A \cap B)$. The new constraint subsumes both $(x \in A)$ and $(x \in B)$. It is rewritten into NIL when $A \cap B$ equals $\emptyset$.

In both cases, the instantiation of a variable x with a constant a results in the evaluation of the constraint: both $((type\ a) = T_A)$ and $(a \in A)$ can be rewritten into either T (if a satisfies the constraint) or NIL (if a does not satisfy the constraint).

[15]In practice, a disjunction reduced to a single type (e.g. (*or robot*)) is rewritten into the type (e.g. *robot*) and an empty disjunction (*or*) is rewritten into $\emptyset$. A disjunction which includes comparable types is rewritten into a smaller disjunction as one type subsumes the other. For example, (*or robot mobile-robot*) becomes *robot* and (*or robot mobile-robot workstation*) becomes (*or robot workstation*).

Disjunctions of Equalities and Disequalities

The propagation of "object-attribute-value" relations result in the creation of disjunctive constraints. The constraint propagation system can in turn propagate these constraints.

- When a disjunct (e.g. $x = x$) is a tautology, the disjunction is also a tautology and is rewritten into T.
- When a disjunct (e.g. $x \neq y$) follows from another constraint of the constraint base (e.g. $y \neq x$), the constraint subsumes the disjunction.
- When a disjunct (e.g. $x \neq x$) is a contradiction, the disjunct is removed from the disjunction.
- A disjunct (e.g. $x = y$) in contradiction with another constraint of the constraint base (e.g. $y \neq x$) is removed from the disjunction.
- A disjunction reduced to a single disjunct ($or\ D$) is rewritten into D.
- A disjunction with no disjunct is a contradiction and is rewritten into NIL.

An additional possibility of interest (but a priori costly) with respect to these subsumptions and simplifications is to use type or domain constraints to prove that two objects (constants or variables) are necessarily different: when two variables x and y have inconsistent types or disjoint domains, the constraint $(x \neq y)$ allows the subsumption and the simplification of disjunctions; when a constant a does not belong to the type or domain of a variable x, the constraint $(x \neq a)$ allows the subsumption and the simplification of disjunctions.

Note that disjunctive constraints can remain in the constraint base when a plan is complete. This raises the problem of determining whether the set of disjunctive constraints is consistent with the other constraints of the constraint base. As long as disjunctive constraints are not combined one with the other, constraint propagation tells us (at most) that each disjunct is consistent with the non-disjunctive constraints. This does not mean that for each disjunction we can choose a disjunct so that the chosen disjuncts are altogether consistent with the non-disjunctive constraints. In other terms, the constraint propagation steps discussed above do not allow to prove that a constraint base containing several disjunctions is consistent. In theory, the generalized resolution rule mentioned in section 2.2 could be used to propagate disjunctions further and prove the global consistency (or inconsistency). In practice, this would be extremely time-consuming. Instead we let the task planner explore the different possibilities in a depth-first fashion. At each step, a disjunct is selected and added as a new constraint. A solution is found when a disjunct of each disjunction has been selected and added to the constraint base with no contradiction. The number of disjunctive constraints to solve in this manner is usually small. Therefore, the exploration is not too time-consuming.

3.2.2 Constraint Propagation Scenarios

Most of the constraint propagation steps discussed above are necessary for the task planner to work correctly, and/or relatively inexpensive in comparison with the resulting savings in search. On the other hand, a few constraint propagation steps are difficult to evaluate in this respect. This leads to the definition of different constraint propagation scenarios among which a choice is difficult.

The first problem is to decide whether to use type specifications or domain constraints. On the one hand, type specifications allow the construction of situation-independent plan structures. Only drastic changes in the environment can make a set of plan structures incomplete. On the other hand, domain constraints refer to specific agents and objects in the environment. The choice of specific agents and objects to accomplish a given goal is still performed at instantiation time, but situation-specific information about available agents and objects is used to prune the search space at plan construction time. Therefore, the use of domain constraints can prove much more efficient, especially if the environment consists of very dissimilar agents and objects.

The second problem concerns the combination of information about the initial situation with "object-attribute-value" constraints (*in-context I* ((*attribute object*) = *value*)). As explained above, three possibilities are worth considering.

- *Generate disjunctive constraints.* This is the simplest thing to do, but entails the potentially costly propagation and satisfaction of disjunctive constraints.
- *Restrict constraint propagation to cases in which either object or value is a constant.* This generates and propagates less constraints. But important deductions (replacement of variables with constants) are still enabled.
- *Achieve arc-consistency.* This is possible only if domain constraints are associated to *object* and *value* (when *object* and *value* are variables). This makes more constraint propagation than above. But the achievement of arc-consistency is probably less costly than the generation of disjunctive constraints.

The third problem concerns the use of type and domain constraints to subsume and simplify disjunctions. Although this is a priori costly, the subsumption and simplification of disjunctions greatly simplifies the overall constraint satisfaction problem. Without experimentation, it is difficult to determine whether using type and domain constraints to subsume and simplify disjunctions is beneficial or not.

Ten constraint propagation scenarios are consequently considered with respect to these three issues.

- A: use type specifications, generate initial-context disjunctions, use type specifications to subsume and simplify disjunctions.
- B: use type specifications, generate initial-context disjunctions, do not use type specifications to subsume and simplify disjunctions.
- C: use type specifications, limit initial-context propagation to constants, use type specifications to subsume and simplify disjunctions.
- D: use type specifications, limit initial-context propagation to constants, do not use type specifications to subsume and simplify disjunctions.
- E: use domain constraints, generate initial-context disjunctions, use domain constraints to subsume and simplify disjunctions.
- F: use domain constraints, generate initial-context disjunctions, do not use domain constraints to subsume and simplify disjunctions.
- G: use domain constraints, limit initial-context propagation to constants, use domain constraints to subsume and simplify disjunctions.
- H: use domain constraints, limit initial-context propagation to constants, do not use domain constraints to subsume and simplify disjunctions.
- I: use domain constraints, achieve arc-consistency, use domain constraints to subsume and simplify disjunctions.
- J: use domain constraints, achieve arc-consistency, do not use domain constraints to subsume and simplify disjunctions.

Section 3.3 presents experiments made to compare these scenarios. Section 3.4 discusses the results.

3.3 Description of the Experiments

The ten constraint propagation scenarios above have been used to solve four types of planning problems: *motion*, *connection*, *object transportation* and *document reproduction.*

Mobile robots submit *motion* and *connection* goals to themselves: after the completion of a task, a mobile robot plans either to go back to a dedicated room (*motion*) or to connect onto the computer communication network to determine what to do next (*connection*). The corresponding planning problems are very simple. A plan to achieve $((location\ myself) = R)$ consists of a unique *move* action. It contains no variable. A plan to achieve the connection goal ((*on-line myself*) = T) consists of two actions: *move* and *connect.* It contains one variable: the communication port with which the robot will connect.

On the other hand, *transportation* and *reproduction* goals are submitted to a central task planner and scheduler (CTPS). The CTPS constructs plan structures for these goals. Then all the available robots instantiate the plan structures — with respect to their situation and knowledge — and the CTPS chooses the best allocation (see section 4). Plan structures for the *transportation* of a given object involve two variables: a mobile robot and its initial location. If the mobile robot and the object are not in the same room in the initial situation, the robot must *move* prior to *get* the object, then *move* again and manage to *leave* the object in the required location.

A *reproduction* goal specifies that a new copy of a document is required in some location. Five different plan structures (with 8 actions and 5 variables on average) are built when constraint propagation is reduced to a minimum. The planner must consider facts such as "there is no mobile copier" to reduce the number of possible plans. More complex facts like "no library contains a copier" allow to prune the search space when domain constraints are in use. Finally, there is a side effect to undo: the exemplar used to make the new copy must be brought back to its original location (see the discussion of unsuitable side effects in section 3.1).

The ten constraint propagation scenarios have been applied to each planning problem ten times (with different initial conditions) in a small environment. The environment is similar to the Robotics Laboratory. It consists of 22 rooms (with 1 corridor) and contains 4 objects of each "most specific" class (4 books, 4 copies, 4 clients, 4 mobile robots, etc) except for copiers (1 copier) and libraries (2 libraries). The two best constraint propagation scenarios (C and I) were selected to run more and more complex *connection* and *reproduction* examples. These examples involve 8, 16, 32 and 64 objects of each class, except for rooms, corridors, copiers and libraries, which were kept unchanged.

Table 1 presents the results for the small environment. Table 2 presents the results for *connection* goals in the more complex environments. Table 3 presents the results for *reproduction* goals in the more complex environments. For each scenario and each problem, the appropriate table provides (a) the time needed to compute all the possible plan structures, (b) the time needed for a given robot to compute all the possible plan instances and choose the best given the possible plan structures and (c) the total (planning + instantiation) time. Each figure in table 1 is the average of ten values. Each figure in table 2 is the average of n values where n is the number of mobile robots. Each figure in table 3 is the average of three values corresponding to the following cases: the new copy is expected in the room where the copier is; the new copy is expected in a library containing an exemplar of the document; the new copy is expected in another room. Time is given in seconds CPU excluding garbage collection.

Scenario / Problem	Motion	Connection	Transportation	Reproduction
A: Planning	0.08	0.43	0.48	8.52
A: Instantiation	0.00	0.25	0.08	7.20
A: Total	0.08	0.68	0.56	15.72
B: Planning	0.08	0.57	0.73	16.65
B: Instantiation	0.00	0.26	0.08	7.90
B: Total	0.08	0.83	0.81	24.55
C: Planning	0.08	0.20	0.51	4.69
C: Instantiation	0.00	0.17	0.24	1.14
C: Total	0.08	0.37	0.75	5.83
D: Planning	0.08	0.19	0.54	6.28
D: Instantiation	0.00	0.17	0.18	1.84
D: Total	0.08	0.36	0.72	8.12
E: Planning	0.11	0.48	0.66	6.53
E: Instantiation	0.00	0.03	0.09	0.25
E: Total	0.11	0.51	0.75	6.78
F: Planning	0.10	0.63	0.57	13.28
F: Instantiation	0.00	0.03	0.10	0.28
F: Total	0.10	0.66	0.67	13.56
G: Planning	0.08	0.23	0.61	6.25
G: Instantiation	0.00	0.15	0.25	0.58
G: Total	0.08	0.38	0.86	6.83
H: Planning	0.08	0.24	0.57	8.76
H: Instantiation	0.00	0.20	0.26	0.62
H: Total	0.08	0.44	0.83	9.38
I: Planning	0.08	0.24	0.63	3.38
I: Instantiation	0.00	0.17	0.13	0.39
I: Total	0.08	0.41	0.76	3.77
J: Planning	0.08	0.27	0.66	6.97
J: Instantiation	0.00	0.18	0.14	0.41
J: Total	0.08	0.45	0.80	7.38

Table 1: Experiments With Ten Propagation Scenarios

	4 objects per class	8 objects per class	16 objects per class	32 objects per class	64 objects per class
C: Planning	0.20	0.22	0.23	0.25	0.30
C: Instantiation	0.17	0.33	0.66	1.31	2.52
C: Total	0.37	0.55	0.89	1.56	2.82
I: Planning	0.24	0.27	0.30	0.38	0.58
I: Instantiation	0.17	0.35	0.73	1.56	3.17
I: Total	0.41	0.62	1.03	1.94	3.75

Table 2: Planning Connections in More Complex Environments

	4 objects per class	8 objects per class	16 objects per class	32 objects per class	64 objects per class
C: Planning	4.69	4.87	4.91	4.96	5.02
C: Instantiation	1.14	1.99	3.73	7.07	12.59
C: Total	5.83	6.86	8.64	12.03	17.61
I: Planning	3.38	3.91	4.53	6.28	9.70
I: Instantiation	0.39	0.65	1.17	2.20	4.30
I: Total	3.77	4.56	5.70	8.48	14.00

Table 3: Planning Reproductions in More Complex Environments

3.4 Discussion of the Results

In most cases, the use of type and domain constraints to subsume and simplify disjunctions is beneficial. The most complex *reproduction* example is the most interesting in this respect. In some cases, the planner saves 50% of the time needed to construct the plan structures. Instantiation time is more stable because in general instantiation does not result in the generation of new disjunctive constraints.

On the other hand, the propagation of "initial-context disjunctions" takes an important amount of time. When type constraints are in use (scenarios A and B), the time spent during the construction of plan structures does not even result in a prompter instantiation. The disjunctive constraints generated at plan construction time are not used at that time to rule out impossible instances. The situation is different when domain constraints are in use (scenarios E and F): constraint propagation allows to remove impossible values from the domains and a smaller number of constraints remain at instantiation time. The achievement of arc-consistency (scenarios I and J) appears as a good intermediate solution. Instantiation time is smaller than when "initial-context propagation" is reduced to relations involving constants (scenarios G and H), and planning time remains quite small in comparison to the planning time of scenarios E and F. Interestingly enough, scenarios I and J perform better than scenarios G and H both at planning and at instantiation time. The achievement of arc-consistency allows to prune the search space at planning time without sustaining the cost of propagating "initial-context disjunctions".

The most difficult problem is to decide whether to use type specifications or domain constraints. The planning time of scenario C does not grow much with the size of the environment. This is normal since the propagation and planning processes are in this case (as well as for scenario D) situation-independent. Tests regarding constants (e.g. "myself") can become more time-consuming but **most of the propagation steps performed during plan construction do not change with the size of the overall domain.** Instantiation time increases in a much more regular fashion. In particular, the evolution of instantiation time on the *connection* example is simple to interpret: when the number of values for a unique variable doubles, the time needed to compare all the possible instantiations doubles.

The evolution is different when scenario I is in use. Both planning and instantiation time increase with the size of the environment. On the *reproduction* example, the planning time with scenario I becomes bigger than the planning time with scenario C. But scenario I remains the most rapid because the instantiation time remains much smaller for scenario I than for scenario C. On the *connection* example, scenario C is the best even in the simplest environment. It remains the best as the environment grows.

An important remark is that the total (planning + instantiation) time provided in the tables corresponds to one instantiation. In practice, the plan structures obtained with scenario I remain usable as long as no new object (e.g. no new robot) is introduced in the environment. The same plan structures are consequently used a number of times. The plan structures obtained with scenario C remain usable as long as no new type of object (e.g. no mobile copier) is introduced in the environment. The same plan structures are consequently used an even higher number of times. The choice between type specifications and domain constraints is difficult to make without discussing what is stable and what is not stable in the considered environment.

One must always consider experimental results with caution. The results discussed in this section correspond to a particular planner, a particular implementation and a particular set of instantiation heuristics:

- As mentioned in section 3.1, the current task planner operates under the "linearity assumption". It is difficult to know whether the most efficient constraint propagation scenarios will remain the most efficient for extensions enabling the resolution of non-linear problems (e.g. consider contexts as variables and generate more complex disjunctive constraints to account for their possible relations).

- Given an application and a constraint propagation scenario, one can spend more programming time to make the constraint propagation process more efficient. For example, one can consider the use of bit vectors to accelerate either (a) the intersection of domains if domain constraints are chosen and the overall domain is stable or (b) the computation of greatest lower bounds in the semilattice if type specifications are chosen and the hierarchy of types is stable (see [3]). Depending on the application, the effects of these improvements could be more important for a scenario than for another.

- The results are also relative to the heuristics used during plan instantiation. Constraint propagation does not only allow to prune the search space. It also provides information (e.g. number of possible values for a variable) allowing to heuristically organize the exploration of the search space. "Graceful retreat" (most critical task first) and "least impact" (least critical resource first) [65] [75] are examples of heuristics which are better applied when an appropriate amount of constraint propagation is performed. In some cases, a minimal amount of constraint propagation is even necessary to use specific heuristics. The dilemma is not "more constraint propagation" versus "less constraint propagation". It is "more constraint propagation and the opportunity to use the heuristics" versus "less constraint propagation and no (or reduced) opportunity to use the heuristics". The default **domain-independent** instantiation heuristics of the task planner (see section 3.1) have been applied for all the constraint propagation scenarios. In some domains, the existence of efficient **domain-specific** heuristics could eventually make a constraint propagation scenario more interesting than another.

4 Experiments With an Incremental Task Allocation System

The distinction between the generation and the instantiation of plan structures allows several robots to instantiate the same plan structures in parallel. Each robot can therefore determine the best instantiation in which it could appear given its own knowledge and its current situation. This is much more efficient than communicating all this information to a central system which would perform the different computations in a sequential fashion. This does not mean a central system is useless. When several tasks are pending and several robots are available, a central system (provided with a more global view of the environment) can be helpful in deciding which robot performs which task.

This section presents a framework allowing to compare task allocation methods using constraint propagation. Section 4.1 presents the task allocation framework. Section 4.2 presents experiments made to compare a few task allocation methods. Section 4.3 discusses the results.

4.1 A Framework for Incremental Task Allocation

When several robots are available to contribute to achieve goals of other agents, a partially centralized method can be used to determine an appropriate task allocation. Each available robot communicates with a central task planner and scheduler (CTPS). The CTPS cannot communicate with un-connected robots. However, most of these robots picked tasks in accordance with the CTPS. The CTPS approximately knows what these robots are doing. Generally speaking, the CTPS has a global view of (a) the tasks to be performed in the environment and (b) the availability of robots to perform these tasks.

The task allocation framework is reminiscent of the contract net problem-solving formalism (see for example [129] and [109]). When the CTPS receives orders (goals) from clients, it generates plan structures enabling the achievement of these goals and provides available robots with a description of pending goals and plan structures.[16] Robots determine which roles they can play in the execution of the plan structures. This means each robot substitutes itself to variables the type of which is consistent with its own type and propagates the substitutions to check the satisfiability of the other constraints. Then each robot uses the constraint satisfaction algorithm presented in section 3.1 to extend the substitutions and propose complete substitutions to the CTPS. Each proposal includes (a) a reference to the considered goal, (b) a reference to the considered plan structure, (c) a description of the

[16]The CTPS can do this because plan structures do not depend on the particular agents contributing to the satisfaction of goals. Conversely, the CTPS cannot instantiate plan structures since it does not know which agent is going to be available to do what.

substitution and (d) values returned by evaluation functions considered during constraint satisfaction (e.g. an estimate of the total execution time).

The CTPS serves as an intermediate between robots. It ensures that two different robots will not start executing the same task. Two possibilities are distinguished: the CTPS can either make task allocation decisions or organize negotiations among robots. The result (given a set of proposals and a set of goals) is for each goal either to wait or to allocate the goal to a given robot. The process is incremental. Decisions are revised each time a new proposal arrives or when an abnormal situation such as the unexpected absence of a robot is detected. For example, when the CTPS makes task allocation decisions and a new proposal arrives, the CTPS explores solutions including the new proposal to determine whether the current allocation can be enhanced. When there is no more proposal to review, the CTPS asks robots to execute plans with respect to the chosen allocation. Robots to which tasks are assigned respond with an acknowledgement and execute the corresponding actions in a distributed fashion.[17] Allocation decisions corresponding to plans being executed can no longer be revised.

4.2 Description of the Experiments

Different task allocation methods are usable in the framework above. Two questions are important:

- Does the CTPS make decisions or does it serve merely as a communication medium ?
- How detailed is the analysis of task interactions ?

From a theoretical point of view, the best allocation is found when the CTPS performs a perfect analysis of task interactions and makes the allocation decisions with respect to the results of this analysis. From a practical point of view, the global optimization problem is NP-complete and the amount of knowledge required to make a perfect analysis is such that the CTPS will never have this knowledge. The design of an appropriate allocation method is consequently a difficult task.

Preliminary experiments have been made to compare a few task allocation methods in an abstract environment with two types of tasks: (a) transportation tasks and (b) tasks requiring both transportation and the use of a fixed manipulator robot. Three parameters are used to change the characteristics of the environment: the importance of manipulation m (i.e. the portion of execution time spent in manipulation given the repartition of tasks and assuming a uniform repartition of initial situations); the number of mobile robots n; and the global load l defined as the product of the task rate and the average time needed to execute each task. The manipulator is unique and the relation $l < min(n\ 1/m)$ is required

[17]There is no synchronization technique similar to the one presented in [52]. Heuristic rules similar to Ow and Morton's early/tardy heuristics [108] are used to order actions requiring the use of the same resources.

to hold (otherwise the number of pending tasks and the average response time between the arrival and the accomplishment of a task might grow toward infinity). Six values of $(m\ n\ l)$ are considered: (0.25 4 2), (0.25 4 3), (0.25 10 2), (0.25 10 3), (0.10 4 2) and (0.10 4 3).

The simplest task allocation methods must at least guarantee that at most one proposal is accepted for each task and for each robot.[18] When the CTPS makes allocation decisions, its search for an optimal set of proposals includes constraint propagation steps allowing to eliminate a proposal as soon as another proposal for the same task or the same robot is in the set of chosen proposals. When robots make allocation decisions, constraint propagation allows the CTPS to determine whether a new proposal is compatible with the proposals robots have chosen during previous negotiations. If there is no conflict, the new proposal is added to the set of chosen proposals. Otherwise, detected conflicts are sent to involved robots and robots decide either to maintain or to change the current allocation.

More complex methods consist in considering task interactions when evaluating admissible sets of proposals. Admissible sets of proposals remain the same, but the criterion to choose among them is more accurate. Let us for example suppose that we want to minimize the average response time between the arrival and the accomplishment of a task. Temporal constraint propagation allows to determine the total time spent waiting for the manipulator when several tasks require its use. As a result, the average response time resulting from the choice of a set of proposals is computed with more precision.

Four task allocation methods have been implemented for the environment described above:

- A: centralized allocation decisions without temporal constraint propagation.
- B: centralized allocation decisions with temporal constraint propagation.
- C: one-to-one negotiation without temporal constraint propagation.
- D: one-to-one negotiation with temporal constraint propagation.

For each task allocation method and each value of $(m\ n\ l)$, table 4 provides:

- the average response time between the arrival and the accomplishment of a task, excluding computational time (to make the result independent of the computing speed);
- the average time per task spent making computations (task planning + plan instantiation + motion planning + task allocation);
- the average time per task spent in task allocation.

[18]The situation of a robot changes when it executes a task. Therefore, it does not make sense to accept two proposals from the same robot. After the completion of a task, a robot must submit new proposals for the remaining tasks.

Method / (m n l)	(0.25 4 2)	(0.25 4 3)	(0.25 10 2)	(0.25 10 3)	(0.10 4 2)	(0.10 4 3)
A: Response Time	1.46	1.61	1.33	1.36	0.87	1.11
A: Computation	15.17	14.82	15.09	17.33	15.65	15.55
A: Allocation	0.009	0.002	0.011	0.015	0.007	0.004
B: Response Time	1.27	1.57	1.26	1.33	0.87	1.08
B: Computation	15.90	13.98	16.06	17.29	16.69	15.16
B: Allocation	0.033	0.088	0.097	0.362	0.041	0.025
C: Response Time	1.54	1.74	1.40	1.49	0.97	1.11
C: Computation	15.13	16.38	18.52	16.05	15.44	15.73
C: Allocation	0.006	0.006	0.013	0.011	0.006	0.004
D: Response Time	1.27	1.58	1.26	1.33	0.87	1.11
D: Computation	15.55	14.50	15.38	16.57	15.32	15.44
D: Allocation	0.029	0.043	0.109	0.140	0.044	0.026

Table 4: Experiments With Four Task Allocation Methods

The unit chosen for the average response time is the average execution time needed to accomplish a unique task (assuming a uniform repartition of initial situations). This is consistent with the definition of the parameter m. Three important factors influence the average response time: the exploitation of "better-than-average" initial situations (e.g. when a robot is close to an object to transport); the waiting time between the reception of a task and the beginning of execution; the time spent waiting for the manipulator (when there is a queue at the manipulator). Computational times are in seconds CPU. When several robots are making computations in parallel (e.g. building proposals for the same tasks), the time taken into account is the time between the beginning of the first computation and the end of the last computation. The time spent "in task allocation" is the time spent in making allocation decisions. It does not include the time spent constructing proposals (plan instantiation).

4.3 Discussion of the Results

The most surprising fact about these results is that the average response time obtained with method D is always identical (or almost identical) to the average response time obtained with method B. This is not the case for methods A and C. In other terms, the step-by-step resolution of temporal constraints through one-to-one negotiation provides results as good as the global resolution of temporal constraints: the negotiation process leads "almost always" to the global optimum. But when the use of the manipulator in time is not taken into account, global optimization performs better than one-to-one negotiation: negotiation often leads to a local optimum different from the global optimum.

Another important remark is that the CPU time spent in the allocation process remains negligible when compared to the total time spent making computations. The consideration of temporal constraints makes it an order of magnitude larger (25 to 362 microseconds against

2 to 15 microseconds) but this is still small compared to the 15 seconds spent in the other planning activities.

Additional experiments are needed to determine how these results would evolve with the size of the problem, i.e. with the number of mobile robots, with the number of manipulators, and with additional types of tasks. The current centralized allocation methods have an exponential computational complexity: at some point, the time spent searching the optimal allocation will start growing fast. On the other hand, one-to-one negotiation could result more and more in sub-optimal allocations when the number of robots grows. An additional concern is whether the allocation time remains negligible when compared to the total computation and/or plan execution time. Optimization is no longer profitable when the time spent in the optimization process is bigger than the savings that result from the optimization.

Let us note that the task allocation framework a priori allows to test other task allocation methods:

- Heuristic methods that provide "acceptable" solutions without requiring optimality in any sense constitute a group of interesting examples.
- Methods considering the spatial aspects of task execution constitute another group. Answering questions like "are two robots going to operate in the same room at the same time ?" and "how much time will be lost in this spatial interaction ?" could lead to a better evaluation of possible allocations.
- Finally, allocation methods respecting a hierarchical organization of the overall robotic system — similar to organizations proposed in the domain of distributed computing systems (not performing physical actions) (see [97]) — are also of interest.

5 Experiments With a Flexible Job-Shop Scheduling System

The previous sections (3 and 4) gave examples of controlling constraint propagation in the context of two projects concerning the activities of multiple agents either in a building or on a construction site. Sections 5 and 6 discuss similar examples in the manufacturing domain. Section 5 discusses joint work with Anne Collinot and Gérard Pinoteau on a flexible scheduling system integrating predictive and reactive capabilities. Section 6 discusses joint work with Karl Kempf, Naiping Keng and Stephen Smith on an architecture allowing a predictive scheduler and a reactive dispatcher to run in parallel and deal with environmental uncertainty in a consistent fashion.

5.1 A Flexible Job-Shop Scheduling System

Important features of job-shop scheduling problems vary from one shop to another. In the same shop, they also vary from one situation to another. For example, the variation of the duration of operations depends on the manufactured products, and the importance of bottleneck resources varies with the global load of the shop. Consequently, it is necessary to choose relevant scheduling strategies with respect to the problem-solving context. In this section, we show how we have used a knowledge-based scheduling system to test and compare various scheduling strategies in various circumstances.

SONIA [26] [31] is a knowledge-based job-shop scheduling system designed to detect and react to inconsistencies between a schedule and the actual events on a shop floor. The system is built on a variant of the BB1 blackboard architecture [27]. It is provided with both predictive and reactive scheduling knowledge sources which are used to build and modify schedules. Analysing knowledge sources can be employed to evaluate both predictive and reactive problem-solving contexts. Control knowledge sources use analysis results to choose the most appropriate knowledge sources to execute and determine which "behavior" the scheduling knowledge sources should adopt (e.g. which heuristics they should use). A solution maintenance component (built to allow tuning of the trade-off between the anticipation of inconsistencies and the cost of constraint propagation) is used to update schedule descriptions and detect inconsistencies as scheduling knowledge sources make decisions and unexpected events happen on the shop floor.

Two additional knowledge sources are used as interfaces between SONIA and either a shop floor or a simulator. One of them provides the shop floor manager or the simulator with a schedule generated or corrected by SONIA. It is called each time a new (updated) schedule is considered acceptable. The other one informs the solution maintenance component about the actual or simulated course of events.

Workshop	Sheet-Iron Workshop	Bottleneck Area	Hypothetical Workshop
Number of Machines	23	6	12
Number of Tests	15	100	100
Number of Shifts	3.5	2.5	2.5
Number of Operations	100 to 150	30	60

Figure 8: Experiments with Three Workshops

SONIA can be used with a simulation system to investigate various reactive scheduling strategies.

- Given a description of a shop floor, we can measure the utility of reactive scheduling, compared to the use of classical dispatching rules.
- We can investigate the relevance and the efficiency of each knowledge source in various contexts (e.g. over-loaded shop, highly disrupted shop).
- The behavior of scheduling and analysing knowledge sources can be adjusted with respect to several control issues [29]. For example, when a knowledge source fails (some decisions lead to an inconsistency), various strategies from naive chronological backtracking to sophisticated analyses of the failure (enabling the system to avoid further failures and to cancel, not the most recent decision, but the relevant decision) can be considered. The flexibility of the constraint propagation system also enables the adjustment of the amount of propagation performed in evaluating the consequences of both scheduling decisions and actual (or simulated) events.

Section 5.2 describes experiments regarding the use of various backtracking strategies and the control of constraint propagation. Section 5.3 summarizes the results. [25] provides a detailed description of the results and discusses both the global utility of reactive scheduling and the efficiency of reactive methods implemented in the SONIA system.[19]

5.2 Description of the Experiments

Experiments regarding backtracking strategies and constraint propagation have been made for three shop floors: an actual sheet-iron shop (23 machines, 100 to 150 operations per shift), a bottleneck area of this shop (6 machines, about 30 operations per shift) and a

[19]The global utility of reactive scheduling has also been studied by other researchers (e.g. [100] [154]). In addition, an important amount of work regarding the relevance and the efficiency of various predictive and reactive scheduling methods, based on multiple decompositions of scheduling problems, has been done at Carnegie-Mellon University [130] [107].

hypothetical shop obtained by doubling the capacity of this bottleneck area (12 machines, about 60 operations per shift).

Figure 8 shows for each workshop the number of tests made, the number of shifts per test and the number of operations per shift. Shifts are 7 to 12 hours long. The demand/capacity ratio in the sheet-iron shop varies from 30% for under-loaded machines to 90% for bottleneck machines. In the two other shops, machines are equally loaded. The duration of operations varies from a few minutes to a few hours in the sheet-iron shop and from 45 minutes to 3 hours in the two other shops. Tool setups are costly (10 to 40 minutes). Unexpected events are delays, operation interruptions and mainly machine breakdowns. For each machine, breakdowns are generated with respect to a Poisson law the parameter of which is 2000 minutes. In most cases, the machine is unavailable for 90 minutes or less.

Two backtracking issues are considered: (a) intelligent chronological (see section 1.1) or selective backtracking; (b) recording or not recording incompatibilities detected by the solution maintenance component (recording incompatibilities prevents the re-making of the same "mistakes"). Consequently, four backtracking strategies are available.

Similarly, a series of constraint propagation "scenarios" is defined with respect to the following issues:

- Constraint propagation can be restricted to the determination of critical paths in a PERT-like graph [25]. This means disjunctive (resource sharing) constraints are not considered by the constraint propagation system. On the contrary, the constraint propagation system can use the generalized resolution rule (see section 2.2) to confront temporal inequalities with disjunctions (in addition to the determination of critical paths). For example, if a machine is planned to be unavailable throughout the interval of time (6 8), and if an operation *op* must be performed on this machine without interruption, we can deduce $start(op) \geq 8$ from $(or\ (start(op) \geq 8)\ (end(op) \leq 6))$ and $end(op) \geq 7$.

- The solution maintenance component can either ignore or detect re-scheduling opportunities. For example, $end(op) \geq 7$ may disappear in the course of the simulation (because an operation preceding *op* ends earlier than expected, thereby enabling *op* to start earlier than expected). If no other constraint prevents *op* from ending before date 6, the solution maintenance component can then delete $start(op) \geq 8$ and signal an opportunity for re-scheduling *op*.

- We considered two well-known methods for updating critical paths as scheduling decisions are made. One of them consists in exploring the PERT-like graph. Its complexity is $O(n^3)$ in the worst case (where n denotes the number of manufacturing operations considered within a shift). The other one consists in maintaining a matrix M such that $M(i\ j)$ denotes the longest path from node i to node j in the graph. Its complexity is $O(n^2)$ in all cases.

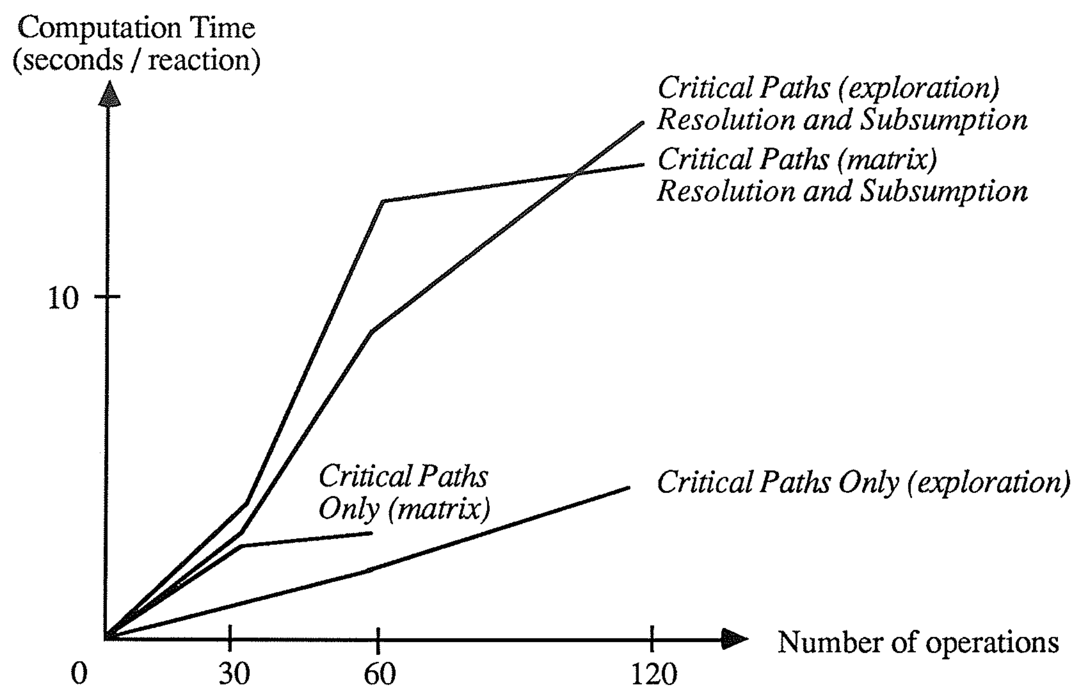

Figure 9: Reaction Time

- When disjunctions of temporal inequalities are considered, we can use subsumption rules to "hide" disjunctions the satisfaction of which results from the satisfaction of other constraints. For example, $(or\ (start(op) \geq 8)\ (end(op) \leq 6))$ can be subsumed by $start(op) \geq 10$.
- For two knowledge sources of the SONIA system, we were able to design appropriate scenarios which were more efficient than others.

5.3 Discussion of the Results

As expected, selective backtracking strategies are in most cases more efficient than chronological backtracking strategies. However, an extended constraint propagation often leads to a significant decrease in the number of backtracks. Consequently, the utility of selective backtracking is reduced (and sometimes reversed) when constraint propagation is extended. Similarly, the recording of incompatibilities is all the more useful as constraint propagation is reduced.

The system reacted more quickly to unexpected events when constraint propagation was restricted to the determination of critical paths (figure 9). Globally, the quality of the schedule finally executed was not significantly altered by the absence (due to reduced propagation) of both early detection of conflicts and detection of scheduling opportunities. On the other hand, for predictive scheduling, the cost of an extended constraint propagation was balanced by a reduction of search in nearly 50% of the cases. In 90% of the cases, the exploration of the PERT-like graph was much more efficient than the matrix-based method (although it is in $O(n^3)$ against $O(n^2)$ in the worst case). The cost of subsuming

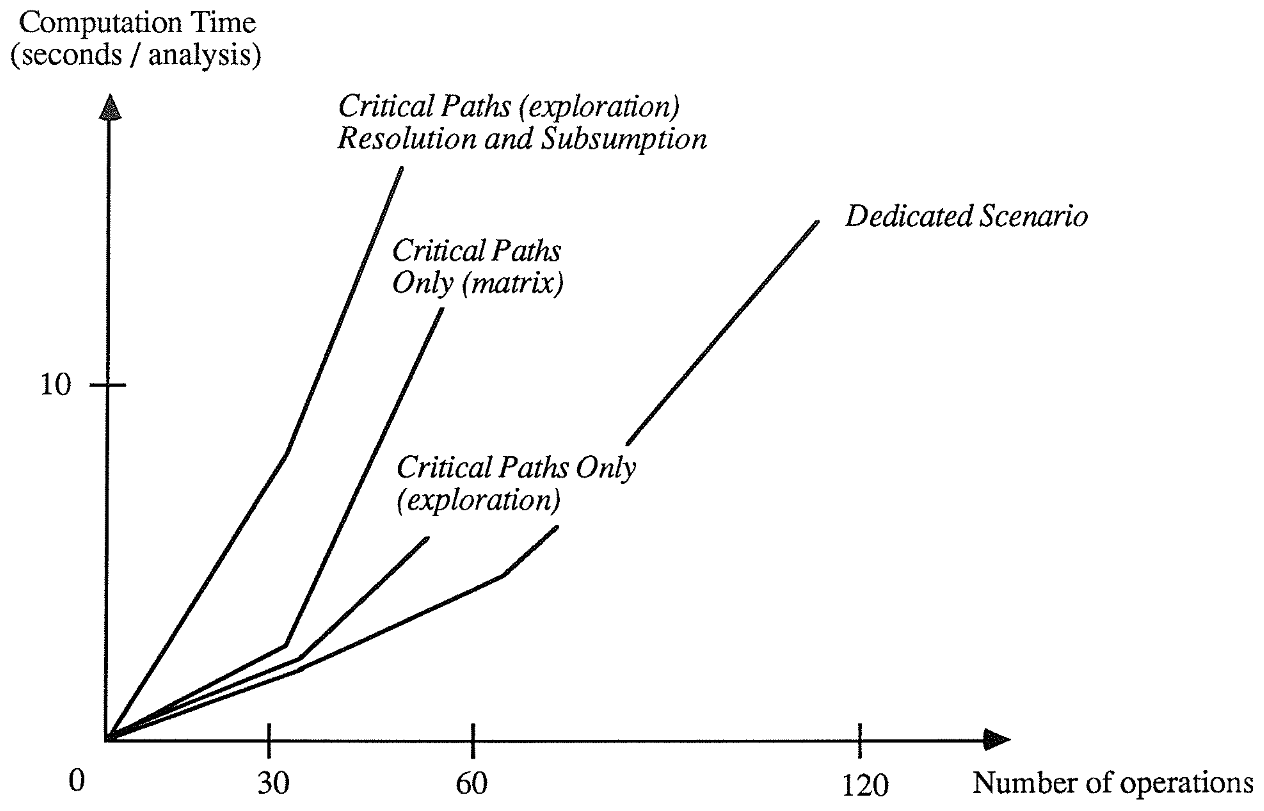

Figure 10: Capacity Analysis

disjunctive constraints was hardly balanced by the resulting reduction of search: in 80% of the cases, subsumption did not enable to save more than 10% of the total (search + constraint propagation) time.

Providing propagation scenarios dedicated to two knowledge sources of the SONIA system turned out to be very worthwhile. By performing reduced propagation, one of them was able to save 18% to 78% of its computational time while producing equivalent results (figure 10). By performing extended propagation, the other one was able to immediately detect cases in which several operations of the same order could not be performed during the same shift without re-scheduling operations of other orders, thereby enabling the whole system to prune the search space.

Results of experiments also show that the suitable amount of constraint propagation varies (in some cases) from predictive to reactive scheduling contexts. This means one must be very careful when one wants to integrate predictive and reactive scheduling components in the same system without reducing the efficiency of predictive and/or reactive scheduling.

6 Relating Scheduling and Execution Monitoring

This section presents an architecture allowing a predictive scheduler and a reactive dispatcher to run in parallel and deal with environmental uncertainty in a consistent fashion. The overall system is a constraint-propagation-based implementation of ideas presented in [133]. It consists of three agents operating in parallel and exchanging messages in an asynchronous fashion:

- The *scheduling* agent generates a predictive schedule and updates it *occasionally* in response to arising problems and opportunities.
- The *execution monitoring* or *dispatching* agent decides in real-time which operations to start on the shop floor given the actual course of events. It relies on the schedule to make its decisions, but overrides it in response to arising problems and opportunities.
- The *interface* agent serves as a mediator between the scheduling agent and the dispatching agent. It determines which agent is allowed to make which changes in which part of the schedule.

Central to this approach is a "distributable" representation of the schedule [133]. We represent the schedule as a set of schedule objects. The most common type of schedule object is a production step.[20] A production step has the basic form (*step job machine st et*), where, for any given production step s, $step(s)$ is the process step to be performed, $job(s)$ is the job being operated on, $machine(s)$ is the machine allocated for the purpose of performing this production step, $st(s)$ is the scheduled start time of the production step, and $et(s)$ is the scheduled end time of the production step. The first two attributes of a production step uniquely identify a particular production activity to be performed. The final three represent the decisions of the scheduling agent. These decisions are made with respect to a number of constraints (e.g. precedence and duration constraints) imported or derived from generic product/process models and made accessible to both the scheduling agent and the dispatching agent. An admissible schedule is a schedule which satisfies all the constraints associated with its schedule objects. It is represented as a graph $(X\ U)$, where X is the set of schedule objects and $U = (U_J \cup U_M)$ is defined as follows:

- $(s_1\ s_2) \in U_J \Longleftrightarrow s_1$ and s_2 concern the same job and s_2 is the immediate successor of s_1 for this job.
- $(s_1\ s_2) \in U_M \Longleftrightarrow s_1$ and s_2 concern the same machine and s_2 is the immediate successor of s_1 for this machine.

Figure 11 presents an admissible schedule and the corresponding graph. We use the J and M marks to distinguish arcs in U_J and U_M. The JM mark is used when the same arc $(s_1\ s_2)$ belongs to both U_J and U_M.

[20]Preventive maintenance and repair operations constitute other types of schedule objects. See [133] for details.

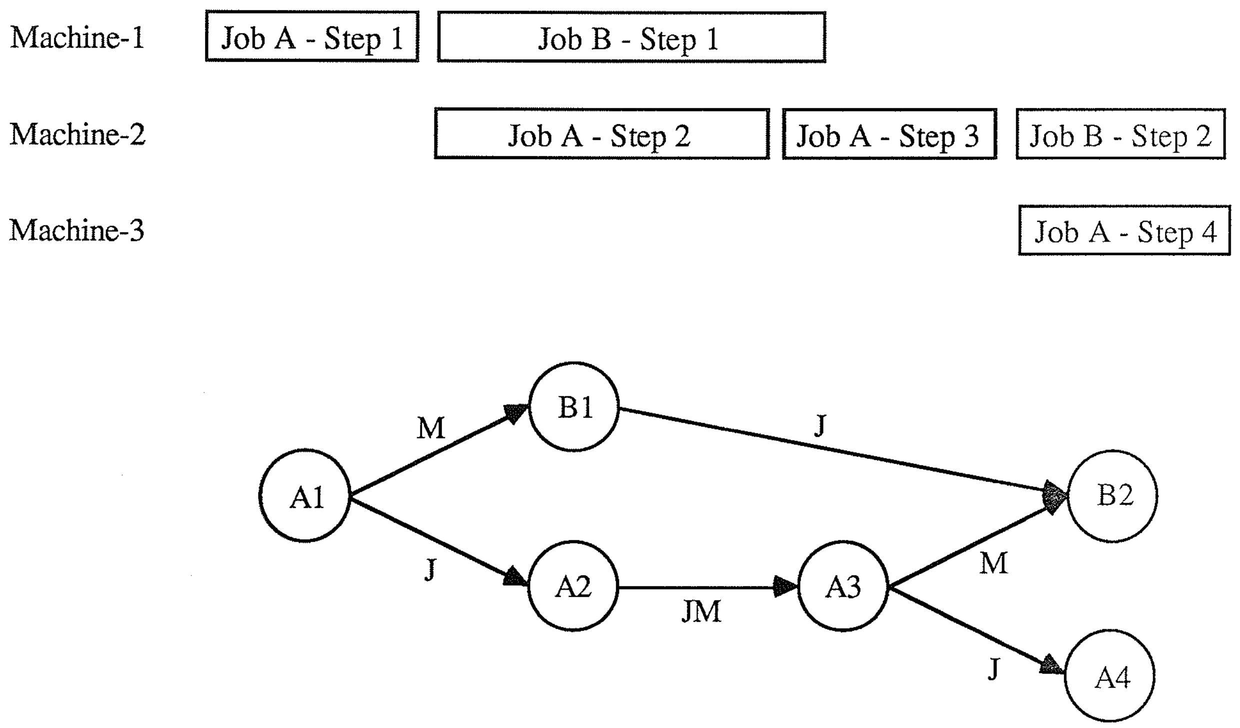

Figure 11: An Admissible Schedule and the Corresponding Graph

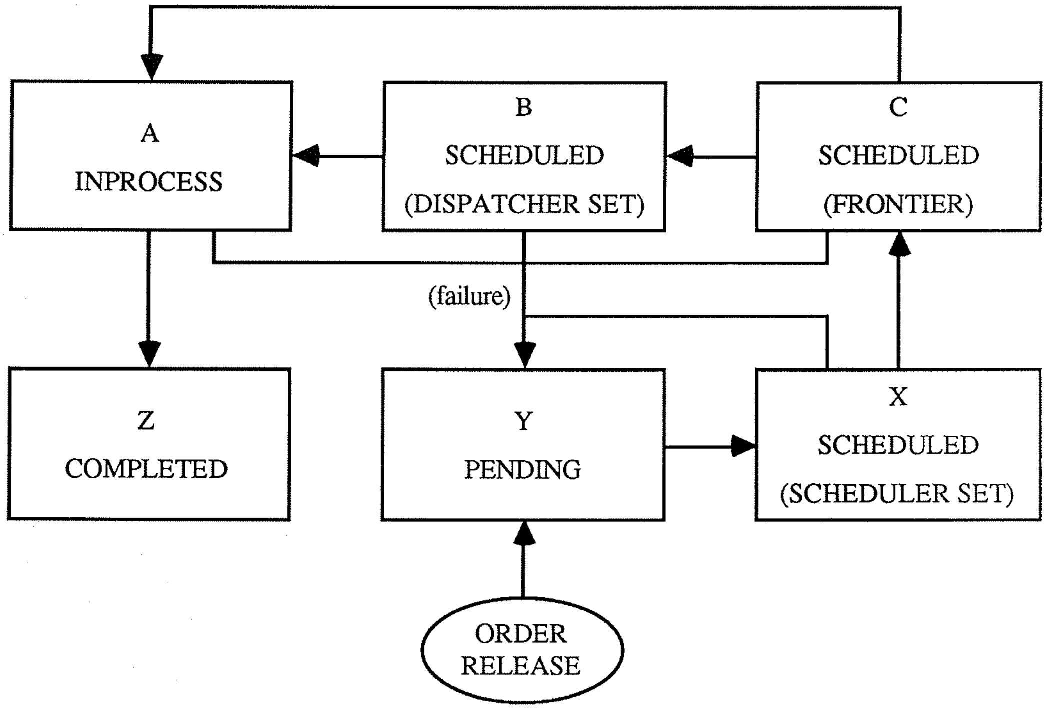

Figure 12: Sets of Scheduled Objects

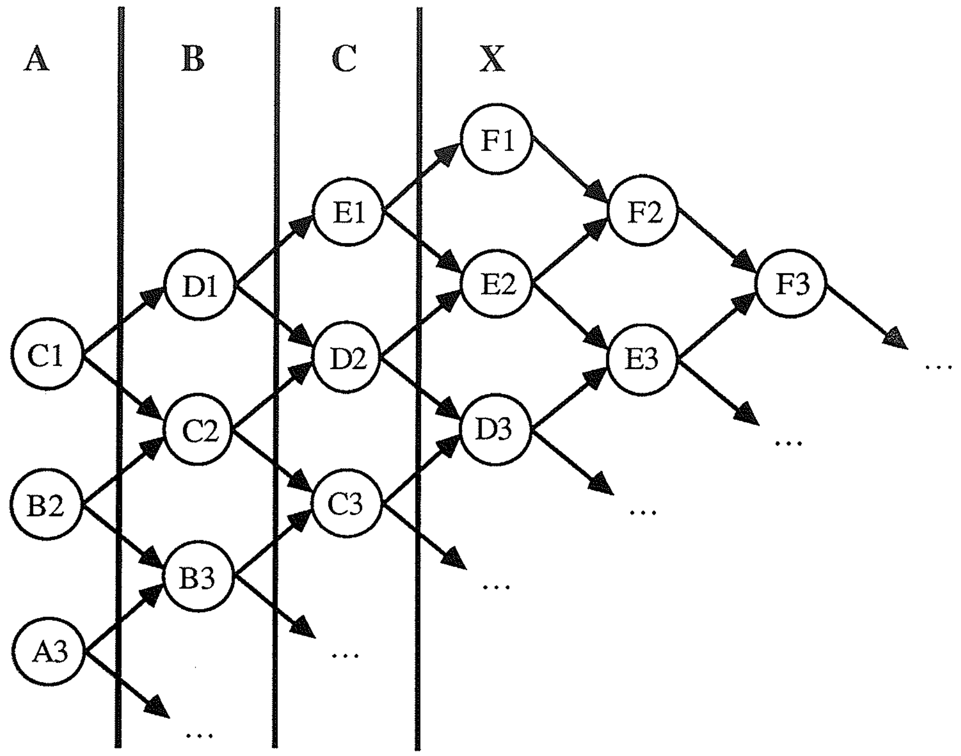

Figure 13: An Initial Situation

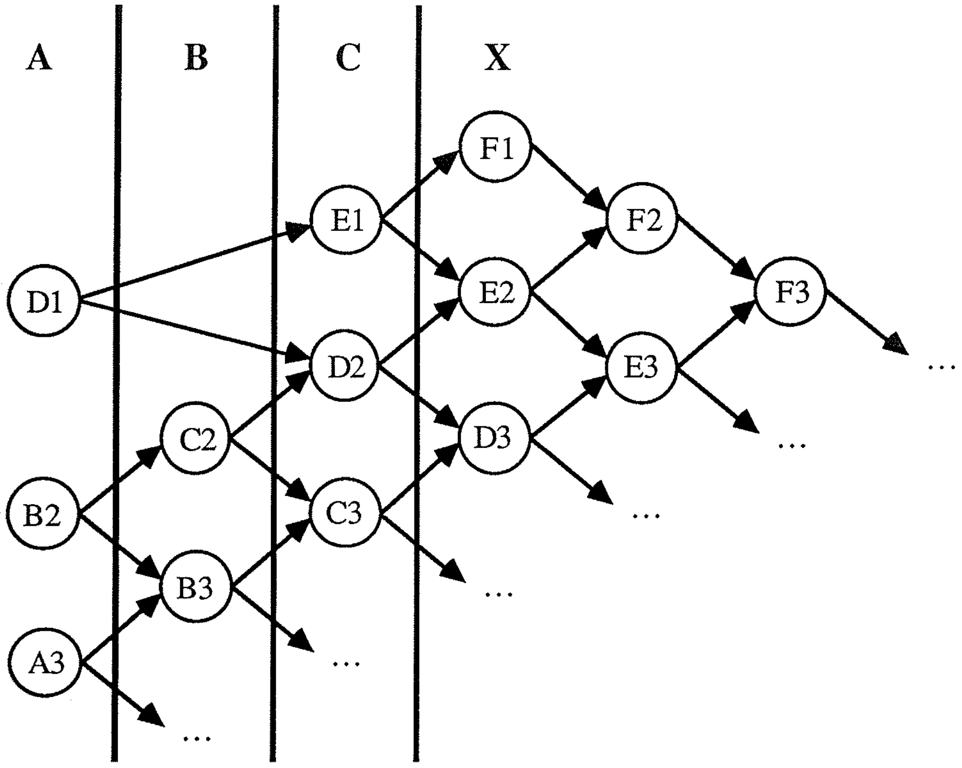

Figure 14: An Execution Decision

To mediate interaction between the scheduler and the dispatcher, the set of all schedule objects is divided into six subsets (as shown in figure 12). Schedule objects are created in response to production and maintenance requests and loaded initially into subset Y. The main task of the scheduler is to move schedule objects from subset Y to subset X where schedule objects have all scheduling decision variables assigned. Within the jurisdiction of the dispatcher, there are three subsets of interest. Subset A contains all the schedule objects that are currently executing. Subsets B and C contain the "close to execute" (a notion to discuss below) according to the schedule: the principal task of the dispatcher is to move schedule objects from subsets B and C to subset A. The schedule objects in C constitute the *frontier* between the jurisdiction of the scheduler and the jurisdiction of the dispatcher. The scheduler cannot touch them and the dispatcher must start them on time. The schedule objects in B are the focus of attention of the dispatcher. The dispatcher is allowed to override the scheduling decision variables in all possible ways provided that it respects the scheduling constraints and the decisions made for the schedule objects in C. When the dispatcher is not able to respect these decisions (or when a schedule object fails to successfully execute), it determines which schedule objects are causing problems. These schedule objects are then returned to Y indicating the necessity of a global schedule repair.

When a schedule object finishes executing, it moves to the last subset, Z, providing a record of the actual behavior of the manufacturing system (as a schedule object s moves from *scheduled* to *inprocess* to *completed*, $st(s)$ and $et(s)$ are updated to reflect the actual times). The dispatcher moves schedule objects from subsets B and C to subset A and requests new schedule objects in replacement. Figures 13 to 15 illustrate this process in a simple case. In this example, we assume that the dispatcher is set to control (a) the executing schedule objects, (b) their immediate successors in the graph and (c) the immediate successors of the immediate successors of the executing schedule objects. Figure 13 shows the initial situation. *C1* (the first production step of job *C*), *B2* (the second production step of job *B*) and *A3* (the third production step of job *A*) are *inprocess*. When *C1* completes, *D1* starts, as shown on figure 14. *D2* and *E1* become the immediate successors of an executing schedule object, so the dispatcher requires successors for them and obtains *D3*, *E2* and *F1* (figure 15). This is the simplest case. In actual fact, three problems can occur:

- The structure of the graph is not that simple. The dispatcher must therefore determine the jobs and the machines for which to require schedule objects. When the dispatcher is set to control n levels of schedule objects (including the *inprocess* level), the number of requests following an execution decision can fluctuate between 0 and 2^{n-1}. In addition, the interface agent must manage to transmit the corresponding schedule objects in an order compatible with the schedule. Figure 16 exemplifies this problem. Even though *C5* is now *inprocess*, transmitting *D5* before its predecessors *D2* to *D4* does not make sense. The interface agent must choose to transmit either *D5* with its predecessors or nothing.

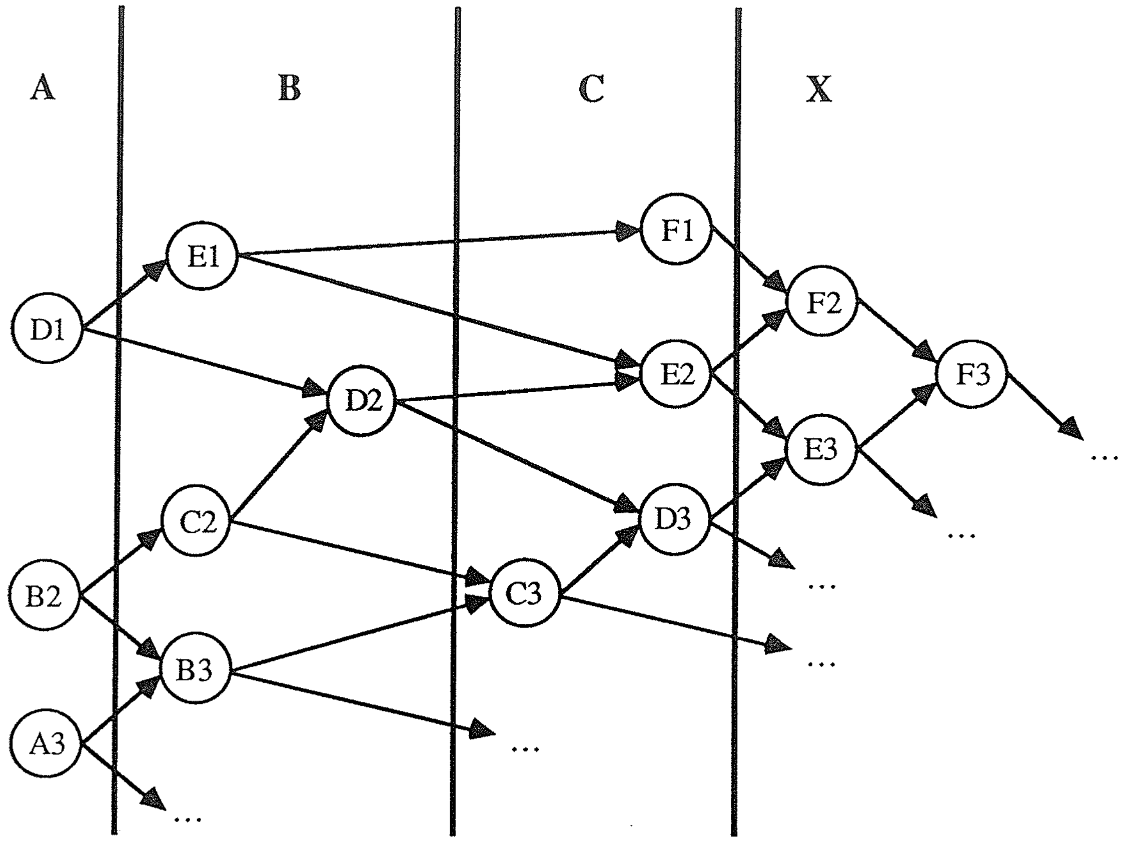

Figure 15: The Transmission of New Schedule Objects

- When the scheduling agent is making changes to the schedule, the interface agent must make sure that it will not transmit a production step under modification. This means the interface agent must know what the scheduler is doing.
- When the dispatching agent is returning schedule objects after a failure, the interface agent must make sure that it will not provide successors of these schedule objects to the dispatching agent. This means the interface agent must know when the dispatcher is about to return schedule objects.

In the current implementation, three precise protocols allow to solve these problems. The *scheduler-interface* protocol applies when the scheduler wants to update the schedule. It consists of four phases:

- The scheduler sends a request to protect a portion P of the schedule ($P \subseteq X$) that it wants to update.
- The interface processes the request. It decides to protect a subset S of P. A message describing S is sent to the scheduler. Then the scheduler knows that the interface will not provide members of S to the dispatcher before the schedule revision is done.
- The scheduler revises the schedule. It sends messages to the interface when it can release schedule objects in S.

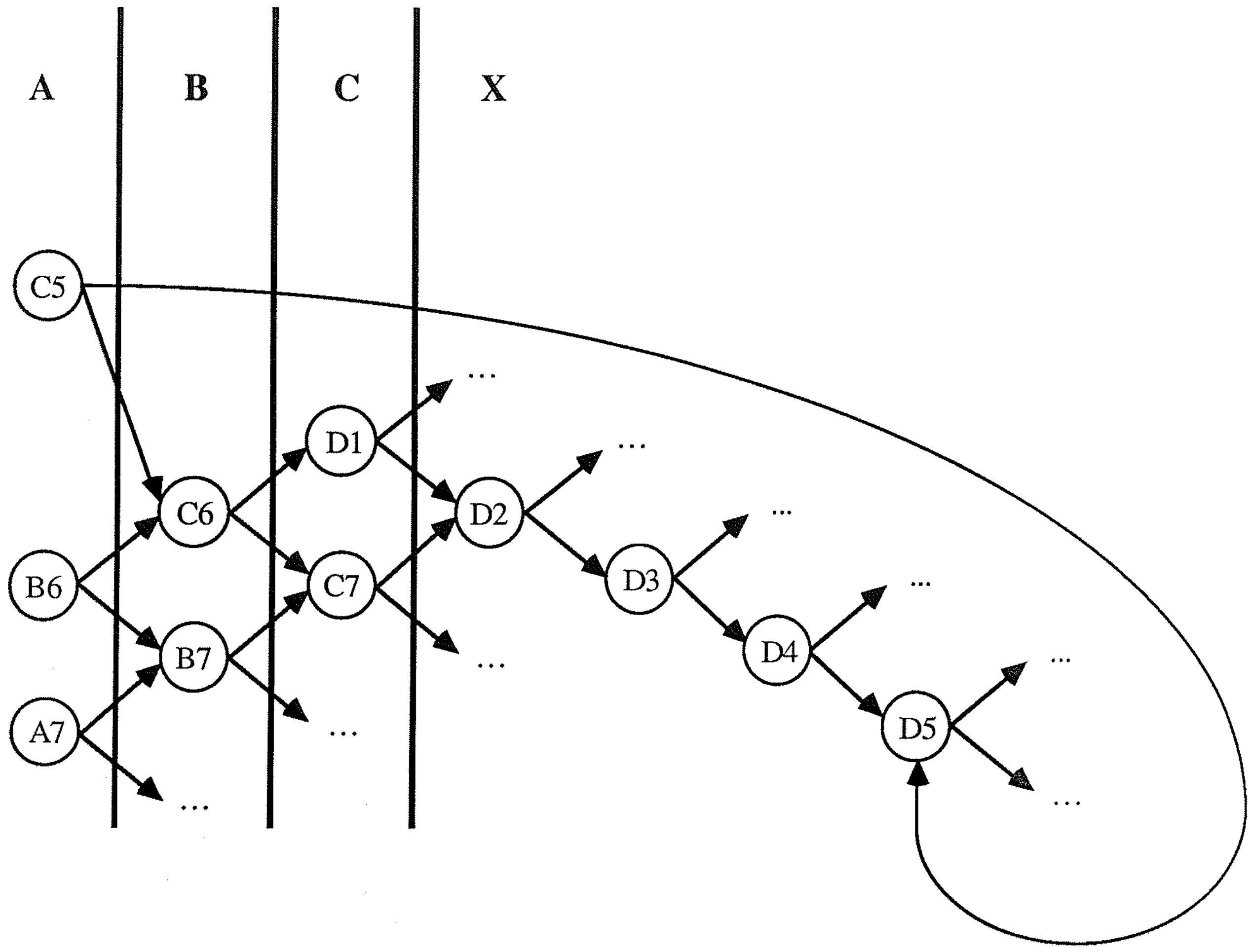

Figure 16: An Ill-Structured Graph

- When it receives release messages, the interface agent notes that the corresponding schedule objects are no longer under protection and determines whether pending requests from the dispatcher are now processible.

In its principles, the protocol allows all kinds of subset descriptions for P and S. But in practice there is a tradeoff between the generality of the subset description language and the speed of the subset protection mechanisms. The current implementation assumes that the scheduler is able to update the schedule in a reasonable amount of time. When this is the case, the interface agent can afford to protect the complete schedule while the scheduler updates it. There is in fact no subset description to manipulate: the scheduler requires the complete schedule X and gets it.[21]

The *dispatcher-interface* protocol applies when the dispatcher needs a schedule object for a machine or a job. It consists of four phases:

- The dispatcher searches its part of the graph to determine how its decisions to start schedule objects on the shop-floor call for new schedule objects. A simple breadth-first search algorithm is sufficient for this purpose. Then the dispatcher sends its requests to the interface. Each request specifies either a job or a machine for which a schedule object is needed.

- In the second phase, the interface agent processes requests and sends schedule objects to the dispatcher. This is the most complex phase as several cases can occur. Let us consider the different cases for a machine request:

  - (a) If the interface agent has sent a schedule object for this machine in the past (the schedule object can have been sent in response to a job request) and did not receive acknowledgement of receipt (see the next phase), it knows that the dispatcher is going to receive (or has received in the meantime) a schedule object for this machine. The interface can then ignore the new request.
  - (b) If the machine is down or (c) the next schedule object on the machine according to X concerns a broken job or (d) the next schedule object on the machine according to X is protected or (e) there is no next schedule object on the machine according to X, then the request is not processible. It will become processible when the scheduler updates the schedule.
  - (f) If none of the previous conditions applies, the interface must determine whether the next schedule object on the machine has predecessors in X and decide (using a heuristic) either to send the schedule object with all its predecessors or to keep the request pending. If the decision is to keep the request pending, the interface must record the reason for the failure in order to reconsider the request when some predecessors are sent to the dispatcher. If the decision is to send the schedule

[21]Note that X can change between the time of the request and the response of the interface (because the interface can provide schedule objects to the dispatcher in the meantime). However, "X" always denotes X.

object with its predecessors, the interface must record the decision (to allow the consideration of case (a) for future requests) and determine whether it enables the reconsideration of pending requests.

The different cases for a job request are symmetric to the above, except for case (e): if there is no next schedule object for a job according to X, but there is one according to Y, then the job request can become processible when the scheduler updates the schedule; if there is no next schedule object for a job according to X and Y, then the job is done and the interface can ignore the request.

- When the dispatcher receives new schedule objects, it incorporates them into its schedule and sends an acknowledgement to the interface. In some cases, the incorporation can call for new requests, sent right after the acknowledgement.

- When the interface receives the acknowledgement, it takes note that the dispatcher has received the schedule objects. The fact that the dispatcher has integrated the schedule objects in its schedule modifies the conditions in which case (a) applies.

An important remark is that this protocol supposes that the interface receives and processes the messages of the dispatcher in the order in which the dispatcher sends them. Depending on the overall system architecture, this can cause problems or not.

The *cleaning* protocol applies when the dispatcher is not able to respect the decisions made for the schedule objects in C or when a schedule object fails to successfully execute. It consists of six phases:

- The dispatcher sends a message to the interface agent mentioning the need for cleaning its schedule. From this point, the dispatcher stops requesting new schedule objects. It will resume requesting schedule objects at the end of the third phase.

- The interface agent receives the message and sends an acknowledgement of receipt to the dispatcher. From this point, the interface agent stops processing requests from the dispatcher. Assuming the interface processes the messages of the dispatcher in the order in which these messages are sent, all the requests have been considered at least once. The interface agent will resume processing these requests at the end of the sixth phase.

- The dispatcher decides which schedule objects to remove from its schedule (possibly using a heuristic) and sends them to the interface agent, together with new requests reflecting the changes made in the B and C sets. From this point, the dispatcher resumes its normal activities: it will request new schedule objects when needed.

- The interface agent forwards the schedule objects to the scheduler.

- The scheduler removes these schedule objects — and their successors with respect to the job links — from its schedule (these schedule objects go back to subset Y) and sends an acknowledgement to the interface agent.
- When the interface agent receives the acknowledgement, it knows that the schedule is correct again and resumes processing the requests emanating from the dispatcher. It also has to process requests emanating from the scheduler as the scheduler wants to re-schedule the schedule objects in Y. Being back to a normal situation, the interface agent is responsible for the arbitration between the two other agents.

It is important to notice that the three protocols above do not refer or depend upon the scheduling techniques in use in the scheduler and the dispatcher. This means one can replace the scheduling agent with another scheduling agent and the dispatching agent with another dispatching agent **without revising the protocols**. The current version integrates a reduced version of the SONIA scheduling system (see section 5) with the simplest possible dispatcher. Using the solution maintenance component of the SONIA system — with an appropriate constraint propagation scenario — the dispatcher updates the earliest and the latest start time of each schedule object under its jurisdiction each time (a) it receives new information from the shop floor or (b) it receives new schedule objects from the interface. It detects a conflict when a schedule object has its earliest start time greater than its latest start time. The solution maintenance component provides the dispatcher with a description of the conflict which allows the dispatcher to return schedule objects that are causing problems.

The simulation system presented in [91] allows to simulate the cognitive actions of the three agents. It also allows to simulate expected and unexpected events occurring on the shop floor, as well as the actions of a client posting new orders with respect to given statistical laws. From an experimental point of view, the overall system allows to evaluate to what extent the scheduler provides fragile (or robust) schedules: the use of the cleaning mechanism is an indication that the dispatcher is not able to use the schedule to make execution decisions consistent enough with the predictions of the scheduler.

Providing a formal definition for plan or schedule "robustness" is a difficult task. Our intuition is that a plan is "robust" when the violation of the assumptions upon which it is built is of no or little consequence. One can however measure consequences of assumption violations along two dimensions: (a) in terms of the negative effect on performance metrics or (b) in terms of the effort required on the part of the prediction/execution system to recover. Robustness is "the ability to satisfy performance requirements predictably in an uncertain environment" and/or "the degree to which a plan or a schedule provides valid guidance over the range of situations that may be encountered at execution time". Despite their perspectival difference, these two definitions (due to Srinivas Narayanan and to Stephen Smith) provide similar properties to the robustness concept:

- Robustness is an issue when (1) there are response time constraints at execution time and (2) there is unpredictability in the environment.

- A "universal" plan mapping each possible situation onto the best possible course of action in this situation is the most robust since it always provides optimal guidance. But in most domains, such a plan is not constructible: there are too many non-similar possible situations to determine in advance how to react to all of them.

- Even though robustness relates to the coverage of different execution states and/or combinations of events, coverage alone does not constitute a practical measure of robustness. There are two reasons for this: (1) different events have different probabilities of occurrence and (2) there are cases in which the negative effect on performance metrics and/or the revision effort required to recover is small even though it is not zero. From a decision-theoretic point of view, this remark suggests that the consideration of robustness is a compensation to the approximate nature of the performance metrics chosen to evaluate plans and schedules. In most cases, no one knows how to compute the expected value of the actual utility function u, so the planning system is set to optimize an explicit or an implicit function f which approximates the value of u for some probable or non-extreme scenario. Then the notion of robustness appears to denote the degree to which we can expect the execution of a similar scenario and/or the obtainment of a similar outcome.

In the following, we consider the average number of times a schedule object returns from the jurisdiction of the dispatcher to the jurisdiction of the scheduler as a measure of schedule robustness with respect to guidance. To evaluate the robustness of a schedule with respect to performance, we use two classical metrics, average tardiness, and average work-in-process time (WIP time). Let us note that, given a schedule, execution results can vary, not only with the environmental uncertainty, but also with the possibilities made available to the dispatcher. In particular, the criterion determining which schedule objects fall under the jurisdiction of the dispatcher (what is "close to execution") has considerable influence on its ability to move schedule objects in time without "breaking" the schedule.

Up to now, only a small series of experiments has been carried out. In this series, each experimental condition is determined by two parameters N and R. N is the number of levels of schedule objects that the dispatcher is set to control. R is a rule allowing to distribute job slack and machine idle time in the schedule. The *slack time* between two schedule objects s_1 and s_2 is defined whenever $(s_1\ s_2) \in U_J$ as the difference between the start time of s_2 and the end time of s_1. The *idle time* between two schedule objects s_1 and s_2 is defined whenever $(s_1\ s_2) \in U_M$ as the difference between the start time of s_2 and the end time of s_1. Two values (3 and 5) are considered for N. Four values (EB LB EA LA) are considered for R.

- The EARLIEST-BLIND (EB) rule is the one that SONIA uses as a default. SONIA schedules several time periods one after the other. The number and the duration (a few hours to a few days) of these periods are parameters set prior to run the system. The scheduling mechanism assigns a number of production steps to each resource over each period and orders the production steps which cannot execute in parallel. The earliest possible start and end times compatible with this order (and with the assignment of

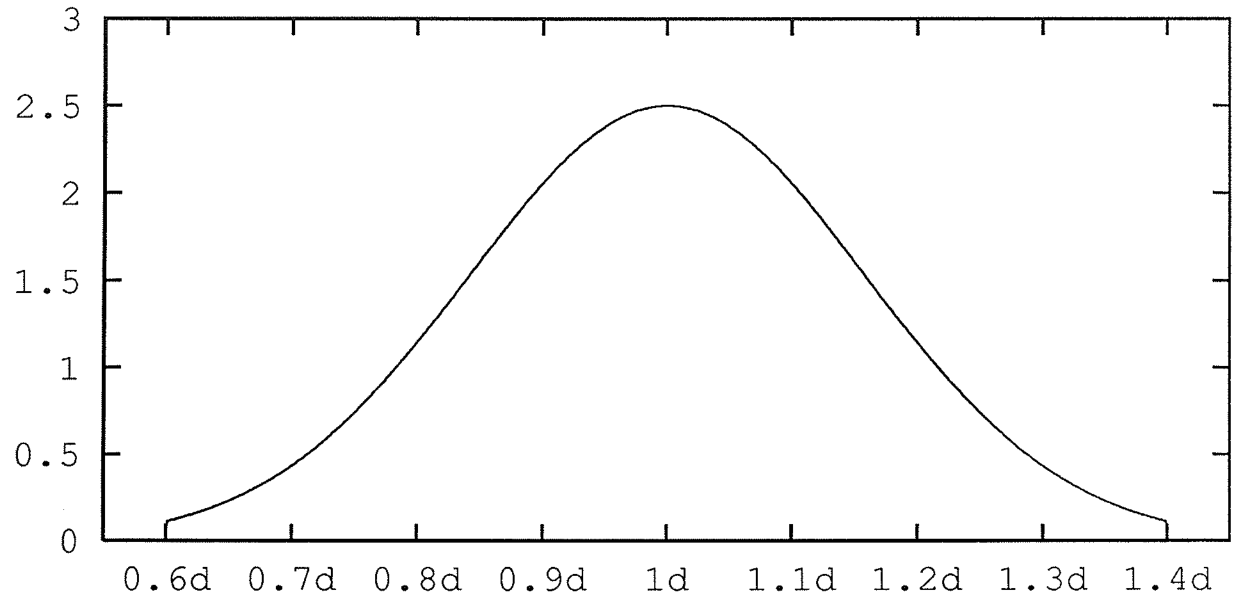

Figure 17: Truncated Normal Distribution for a Schedule Object Duration (d is the average duration)

time periods to production steps) are provided to the dispatcher. This implies that the dispatcher has the smallest conceivable set of possibilities to update the schedule when a problem occurs on the shop-floor.

- The LATEST-BLIND (LB) rule gives more responsabilities to the dispatcher. SONIA provides the dispatcher with the latest possible start and end times compatible with the chosen order and the assignment of production steps to time periods.
- The EARLIEST-AWARE (EA) rule is similar to EB except that the scheduler maintains a "realistic" amount of idle time for each resource over each period. This amount is chosen with respect to statistics gathered on the shop-floor. The scheduler provides the earliest start and end times to the dispatcher: the dispatcher cannot use the additional idle time made available for the ongoing period but it can use the idle time made available at the end of the preceding periods (when this idle time is still available).
- The LATEST-AWARE (LA) rule is similar to LB and EA. The scheduler (a) maintains a "realistic" amount of idle time for each resource over each period and (b) provides the dispatcher with the latest possible start and end times. As a result, the dispatcher can use the additional idle time made available for a period as soon as the first production steps of the period reach the C subset.

Both LB and LA result in a pure propagation of time bound constraints from the scheduler to the dispatcher. The dispatcher can execute each schedule object at any time consistent with the current scheduling decisions. When the dispatcher fails, the scheduler also fails

(N R)	Tardiness (in X)	WIP (in X)	Tardiness (in Z)	WIP (in Z)	Return Rate
(3 EB)	0	415	40	484	1.99
(3 LB)	5	456	2	413	0.54
(3 EA)	6	446	32	471	3.79
(3 LA)	12	484	0	405	0.14
(5 EB)	0	415	17	441	2.95
(5 LB)	5	456	13	430	0.70
(5 EA)	6	446	20	450	3.16
(5 LA)	12	484	14	441	1.71

Table 5: Experiments With Four Slack/Idle Distribution Rules

(and needs to revise its allocation and ordering decisions). In comparison, EB and EA result in the definition of stronger constraints in the jurisdiction of the dispatcher than in the jurisdiction of the scheduler.

Environmental uncertainty concerns the duration of schedule objects. The duration of each schedule object follows a "truncated" normal distribution as shown in figure 17. Given this distribution, a few simulations allow to determine what is a realistic proportion of resource idle time in the schedule. When R belongs to {EA LA}, the scheduler is set to maintain this proportion of idle time in each period. The resulting job slack and resource idle time is then distributed according to the chosen rule R. Table 5 summarizes the results. For each experimental condition, the table provides five figures:

- The average tardiness in the original predictive schedule (interpreted with respect to the chosen rule R).
- The average WIP time in the original predictive schedule (interpreted with respect to the chosen rule R).
- The average tardiness resulting from the schedule execution.
- The average WIP time resulting from the schedule execution.
- The average number of times a schedule object returns from the jurisdiction of the dispatcher to the jurisdiction of the scheduler.

The return rate is (as we could expect) much smaller for rules LB and LA than for rules EB and EA. More interesting is the fact that when N equals 3 this difference in guidance stability results in an important difference with respect to the performance metrics. The overall system is such that the pressure that EB and EA put on the dispatcher results in lots of inefficiencies at execution time.

7 Software Verification in the Planning / Scheduling Domain

In the previous sections (3 to 6), we discussed constraint propagation techniques allowing the "on-line" combination of constraints and the "on-line" detection of conflicts. Given a set of decisions already made by one or several agents, constraint propagation allows to determine whether these decisions are compatible and how they altogether constrain other decisions to come. When decisions are not compatible, a description of the conflicting situation is provided to the conflicting agents, so that the conflicting decisions can be revised.

In this section, we discuss the "off-line" determination of multi-agent system properties and the "off-line" detection of potential conflicts. Given a formal description of the behavior of each agent, we want to determine in which circumstances agents make incompatible decisions and what happens then. This allows to determine whether the overall system is correct (e.g. whether mobile robots avoid collisions, whether important events are responded to on time, whether resource sharing decisions are always made in advance), in which circumstances incidents occur and why they occur. Various methods have been proposed in the literature for solving verification problems of this kind (the most well-known consisting in proving properties of Petri nets [101] which describe the possible behaviors of discrete event systems [112]). We believe constraint propagation techniques constitute an interesting alternative to these methods — in particular when the planning and scheduling activities of the multi-agent system are themselves constraint-driven.[22]

In comparison with traditional software verification problems, an important difficulty in planning and scheduling is the consideration of time. We distinguish three levels of difficulty:

- Theoretically, the simplest case (discussed in section 7.1) is when the effects of actions are not defeasible: any property that becomes true after an action remains true forever after. In this case, a conflict is a situation in which one or several agents try to establish contradictory properties. The resolution of the verification problem does not depend on an explicit consideration of time. This allows the use of techniques developed outside of the planning and scheduling domain such as knowledge base verification techniques if the behavior of agents is described in a declarative fashion.

[22]Here, we will suppose that a mechanism similar to constraint propagation is uniformly used by the agents. This includes the case of rule-based agents since inference engines are describable in constraint propagation terms [87]. Other representations, e.g. Petri nets [101], hierarchies of specialists [33], knowledge sources with declarative bodies [104], may also fit, more or less adequately, in the constraint propagation framework. Most declarative architectures actually consist of a propagation algorithm to determine instances on which system components can apply (all or some instances) and a control language to express situation-dependent preferences among these instances. When the component action parts are also declarative, the overall process is some form of constraint propagation. The representation may change from an architecture to the other, but the basic functions remain the same.

- The simplest case above almost never occurs as an agent can often undo what another agent did. Section 7.2 discusses the case in which the relative ordering of action start and end times influences the evolution of the world and the occurrence of conflicts. In this case, a conflict is a situation in which one or several agents want to maintain contradictory properties over unordered intervals of time.

- Finally, section 7.3 discusses the case in which the continuous evolution of properties (such as the position and orientation of a robot) influences the occurrence of conflicts. In this case, a conflict can occur at any point in time, during the execution of one or several actions. It is no longer possible to focus the attention of the verification system on action start and end times.

Compared to the contents of sections 3 to 6, the discussions below are very prospective. We do not have any well-defined theory of constraint-based software verification in the planning and scheduling domain and we did not make any implementation. Nevertheless, we believe the use of constraint propagation techniques for this task is a possibility to consider.

7.1 Knowledge Base Verification

When the effects of actions are not defeasible, the verification problem consists in determining in which cases a "knowledge base" (corresponding to the overall set of agents) leads to the performance of incompatible actions. In the simplest case, an instance of the verification problem is represented as a triplet (KB IC IF):

- KB is a set of rules ($ANTECEDENT \Rightarrow CONSEQUENT$).

- IC is a set of integrity constraints ($ANTECEDENT \Rightarrow NIL$).

- IF is a set of facts or a way to distinguish facts that cannot be introduced in the working memory of an agent from the outside. For example, a bridge playing system is not supposed to get direct information about which cards are held by which player (except for the dummy player and itself).

The problem is then stated as follows: given the knowledge base KB, the set of integrity constraints IC, and the set of internal facts IF, determine whether the application of rules, from a set of facts that (a) satisfy IC and (b) do not belong to IF, can lead to a violation of integrity constraints (and if yes characterize the origins of the constraint violation).[23] A few remarks follow.

[23]This is only a particular aspect of knowledge base verification (see [103] [96]). The automatic detection of various anomalies (e.g. redundant rules, potential loops and unreachable conclusions) during or after the constitution of a knowledge base is another example of useful assistance. The general idea of using diagnostic meta-expertise to reinforce the quality of a knowledge base is discussed in [111].

- The set of integrity constraints is used both to circumscribe the possible inputs to the multi-agent system and to define the incompatible outputs of the multi-agent system. A more general definition distinguishing two sets of integrity constraints may be needed in some cases.

- The definition does not integrate any constraint on the order in which rules apply. This is adequate if the rules commute. But planning and scheduling agents usually perform some amount of search prior to make a decision. The consequent part of a rule can therefore consist in the deletion of facts. In such a case, the control strategy CS which orders rule applications is important and must be considered as an additional component of the verification problem instance. For example, if the control strategy CS selects rule instances at random (there is no control knowledge), the quadruplet (KB IC IF CS) allows inconsistencies as soon as there is one admissible set of input facts and one application order leading to the violation of IC.

- The complexity of the problem depends on the language available to express rules and integrity constraints. The general problem is NP-hard as soon as the language has the expressive power of propositional logic: if S is a set of clauses $\{C_1 \ldots C_n\}$ and if each clause C_i equals $\{L_{i1} \ldots L_{in_i}\}$, we can construct the rules $(L_{ij} \Rightarrow C_i)$ $(1 \leq i \leq n)$ $(1 \leq j \leq n_i)$ and $(C_1\ AND\ \ldots\ AND\ C_n \Rightarrow S)$ in polynomial time. Then the triplet (KB $\{((NOT\ ?P)\ AND\ ?P \Rightarrow NIL)\ (S \Rightarrow NIL)\}$ $\{C_1 \ldots C_n\ S\}$) allows inconsistencies if and only if S is satisfiable (and the satisfiability problem is NP-complete [51]). The general problem is undecidable as soon as the language has the expressive power of first-order logic: given the description of a Turing machine with its input, we can automatically construct rules that represent the Turing machine, allow the description of the input as the only possible facts coming from the outside and replace the instruction that stops the Turing machine with an integrity constraint. Then there is an inconsistency if and only if the Turing machine can stop (and the halting problem is undecidable [77]).

Many systems have been built to check the consistency of a knowledge base before its use. In most cases, the language considered to express rules is a small extension of propositional logic. For example, Pipard [110] and Rousset [117] accept equalities, disequalities and inequalities between variables and values in the antecedent part, but only equalities in the consequent part. Ayel [7] and Lalo [80] accept first-order predicates at the expense of decidability. The rationale for this choice is simple: partial verification with a first-order language may be more useful than complete verification with a confined propositional language, especially if knowledge exists to focus on the "most likely" potential conflicts, or if decidability is recovered under a few structural restrictions (e.g. no "loops" in [80]). From our point of view, the most definite advantage of the approach advocated in [80] is the possibility of controlling the logical process that leads to the discovery of potential conflicts.

Most systems are also limited to monotonic reasoning, even though an extension to non-monotonic reasoning is in some cases (as in [7]) practicable. Ginsberg [57] and Rousset [117] present simple techniques to handle default reasoning. Ginsberg's system computes when a default hypothesis holds and uses the result as a formula of facts from which the default is "deduced". The drawback of this approach is that the knowledge base must be acyclic in a strong sense (see [57] for details). Rousset's system uses a more complex stratagem. It pretends that the user of the knowledge base can assert the fact "the value of the variable $?V$ is not deducible" and dynamically generates integrity constraints to ensure that this fact cannot coexist with facts allowing to deduce the value of $?V$.[24] This is neat but still insufficient for our purpose. A viable generalization of these techniques is once again to allow the systematic control of the process that leads to the discovery of potential conflicts.

Let us start from the simplest possible case (monotonic reasoning in propositional logic) and examine how a controllable constraint propagation system can generalize to more complex cases. In the monotonic propositional case, one can rewrite rules and integrity constraints in clausal form and use the resolution rule to derive new clauses corresponding to possible sources of conflicts: each clause $\{L_1 \ldots L_n\}$ corresponds to a possible conflict involving $(NOT\ L_1) \ldots (NOT\ L_n)$. The resolution process must continue as long as (at least) one $(NOT\ L_i)$ is an internal fact. Otherwise $\{(NOT\ L_1) \ldots (NOT\ L_n)\}$ is an admissible input which generates a contradiction on the output. Of course the verification system must not generate all the possible clauses. This is where the control issue becomes important:

- A rule and a clause representing a possible source of conflict must admit the rule conclusion as the resolvent. For example, the rule $((NOT\ A) \Rightarrow B)$ and the constraint $(B\ AND\ C \Rightarrow NIL)$ are allowed to combine and provide the clause $\{A\ (NOT\ C)\}$ to represent $((NOT\ A)\ AND\ C \Rightarrow NIL)$. But the same rule $((NOT\ A) \Rightarrow B)$ and the constraint $(A\ AND\ C \Rightarrow NIL)$ are not allowed to combine.

- A big problem in most knowledge base verification systems is the recognition that a set of facts leading to a contradiction violates an integrity constraint (and is therefore of no interest). Here the clause which corresponds to the integrity constraint subsumes the clause which corresponds to the set of incompatible facts. Controlling the constraint propagation system so that it will attempt subsumption before doing anything else with a new constraint (including bothering the system user) suffices to solve the problem.

The resulting verification system does not do more than other verification systems. But it presents important advantages.

[24]This fits well with the overall approach advocated in [117]: a potential conflict may correspond either to an error in the knowledge base KB or to some incompleteness of the set of integrity constraints IC. When the verification system detects a potential conflict, it questions the expert (an oracle) to determine whether KB or IC must change. Assuming that the oracle may introduce incorrect rules in KB but not in IC (IC is at worst incomplete) the verification system finally derives a provably consistent knowledge base from its interactions with the oracle. In the case of default reasoning, the oracle is not needed since "the value of $?V$ is not deducible" is a default to remove whenever a set of facts allows to deduce the value of $?V$.

- Additional control knowledge (heuristic or not) allowing to focus the search for potential conflicts is expressible in the form of additional control rules.

- Assuming that the system records data dependencies (which constraint derives from which and which constraint subsumes which), it functions in an incremental fashion. The addition of a new constraint (from KB or IC) triggers new resolution steps; the deletion of a constraint (from KB or IC) results (a) in the deletion of its consequences and (b) in the restoration of the constraints it did subsume (the restoration triggers new resolution steps). As a result some potential conflicts appear and disappear as the user adds and deletes rules and integrity constraints.[25]

- One does not need to change the constraint propagation framework to accommodate languages wider than propositional logic. One needs a constraint propagation theory for the wider language and control knowledge to guide its application. Languages allowing (*object operator value*) formulas (with *operator* in $\{= \leq \geq < > \neq\}$) in antecedent and consequent parts are examples simple to consider. We can expect that more general languages require more complex control (e.g. necessitate the identification of infinite loops), and we know that complete verification becomes impossible at some point.

- One does not need to change the constraint propagation framework to accommodate non-monotonic reasoning. One has to translate the control rules that control the rule application process into control rules that control the constraint propagation process. The goal is to guarantee that the constraint propagation system can detect resolution steps corresponding to situations that cannot occur when the knowledge base is in use. Given a resolution step providing a new clause $\{L_1 \ldots L_n\}$, the verification system must determine (a) whether a rule cancelling the negation of some L_i is applicable when $((NOT\ L_1)\ AND\ \ldots\ AND\ (NOT\ L_n))$ is true and (b) whether there is a guarantee that this rule applies before $((NOT\ L_1)\ AND\ \ldots\ AND\ (NOT\ L_n))$ results in conflicting outputs.

Another way to solve the verification problem with a constraint propagation system is to generate a constraint for each possible fact that does not belong to IF and allow these facts to combine with rules and integrity constraints to generate other facts and contradictions. This corresponds to running the knowledge base on the set of all possible initial facts (see [87] for an inference engine described in constraint propagation terms). Still control rules are needed to distinguish the combinations of interest: the combination of two constraints C_1 and C_2 is useless as long as C_1 and C_2 are not known to derive from compatible facts. The verification system must use data dependencies to control constraint propagation. The overall process is then similar to what happens in an assumption-based truth maintenance system [36]. We can expect this approach to generalize well to non-monotonic cases (e.g. Rousset's trick is the easiest to reproduce) but not to non-propositional languages since it requires a constraint for

[25]Another incremental approach assuming monotonic propositional logic is presented in [9].

each possible input fact. The examination of general resolution strategies allowing to encode truth maintenance systems (see [18] [19] [20]) suggests solutions to this problem: controlling a resolution process among rules allows to do without facts. Then the generalization of the verification system becomes (from a theoretical point of view) possible.

7.2 Sequencing Constraints

When the effects of actions are defeasible, a potential conflict is a situation in which one or several agents want to establish (or maintain) incompatible properties over unordered intervals of time. The verification system must therefore (1) associate temporal variables (points or intervals) with the properties that must precede and/or follow possible actions and (2) determine whether the multi-agent system allows intervals corresponding to incompatible properties to overlap. This means that the verification system must couple a temporal constraint propagation process with a rule combination process similar to those described above: the rule combination process derives possible paths to incompatible outputs; the temporal constraint propagation process determines conditions that such paths must meet for a conflict to occur. The coupling of these two processes is a complex task. But as soon as a correct coupling framework is made available, it becomes possible to assemble a collection of verification systems (for a collection of simple languages) from existing constraint propagation theories.

An important problem occurs when we start considering actions (or natural processes) occurring at particular times or with particular durations between them. In this case, the verification system has to determine whether some agents do react to some events in a specific amount of time. This means the time taken to make decisions enters into consideration. The verification system must use exact knowledge or given assumptions to compute time bounds on decision-making.

7.3 Constraints on Continuous Evolutions

The most difficult case is when the continuous evolution of properties (such as the position and orientation of a robot) influences the occurrence of conflicts. The resolution of the verification problem then supposes the resolution of complex sets of equations involving continuous functions and (for discrete control) series. Only the simplest of these problems are solvable with the current constraint propagation systems. Constraint propagation theories are in general too weak to handle constraints on continuous evolutions.

8 Conclusion

Kowalski's equation "ALGORITHM = LOGIC + CONTROL" [78] is among the most well-known in computer science (even though most computer scientists do not use the distinction between "LOGIC" and "CONTROL" as a general principle to system design). The planning and scheduling applications presented in this report demonstrate how this distinction allows to consider and compare a variety of constraint propagation algorithms for a given application. The constraint propagation algorithms are based on the same deductive activities: constraint combination; rewriting; subsumption; and correction of the effects of constraint combination, rewriting and subsumption when constraints are removable. What differs from an algorithm to the other is in some cases the "LOGIC" (which determines what inferences may be drawn from a set of constraints) and in most cases the "CONTROL" (which determines what inferences must be drawn and in what order they must be drawn).

Some of the experimental results reported in sections 3 to 6 were difficult to predict. Section 3 suggests that situation-independent task planning is not much more complex than situation-dependent task planning. Section 4 shows that, for a class of task allocation problems, one-to-one negotiation performs almost as well as global optimization, provided that enough temporal details are taken into account. Section 5 mentions (among other results presented in [25] and summarized here) that in scheduling the exploration of conjunctive graphs happens to be more efficient than the use of matrix-based methods — even though the exploration is in $O(n^3)$ against $O(n^2)$ in the worst case. Section 6 shows that schedules constrained to predict good results with respect to usual performance metrics are "fragile" and can bring poor execution results in the presence of uncertainty. Specialists in the planning and scheduling domain can make wrong guesses on such issues. A framework allowing to test and compare constraint propagation algorithms is therefore of high practical importance.

In addition, it appears that even for a relatively confined domain (such as job-shop scheduling for a particular product) there is no "best" constraint propagation scenario. Knowledge on the problem or on the problem-solving strategy must be exploited to reduce the computational burden of solution maintenance without sacrificing the deduction of important information. A controllable constraint propagation system facilitates the utilization of this knowledge.

However, the identification of control knowledge and its expression in the current constraint propagation languages do pose important problems. Customizing a controllable constraint propagation system is not an impossible task for a computer science researcher well wonted to describing complex procedures in abstract mathematical terms. But the current constraint propagation systems do not enable a human problem-solver (e.g. responsible for scheduling operations on a construction site) to make his (or her) own applications as efficient as possible without the help of a specialist. An interesting avenue of research is to make the adaptation of a generic constraint propagation system manageable by its users. This includes:

- the design of more convenient languages to (a) define constraint propagation theories, (b) implement constraint generators and retractors in accordance with application concepts and (c) express control knowledge;
- the constitution of a knowledge base allowing to map task characteristics onto control strategies.

Another — more remote — prospect is to provide a problem-solving system with the ability to gain control knowledge from its experience and adapt its own constraint propagation process to the type of planning and scheduling problems it encounters.

References

[1] Ritu Agarwal and Kislaya Prasad. *Enhancing the Group Decision Making Process: An Intelligent Systems Architecture.* Proceedings of the Twenty-Second Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, 1989.

[2] Hassan Aït-Kaci and Roger Nasr. *LOGIN: A Logic Programming Language With Built-In Inheritance.* Journal of Logic Programming, 3(3):185-215, 1986.

[3] Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln and Roger Nasr. *Efficient Implementation of Lattice Operations.* ACM Transactions on Programming Languages and Systems, 11(1):115-146, 1989.

[4] James F. Allen. *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, 26(11):832-843, 1983.

[5] Jean-Michel André, Anne Mouginot and Michel Venet. *A Framework for Organization Monitoring.* Proceedings of the Ninth European Conference on Artificial Intelligence, Stockholm, Sweden, 1990.

[6] Stefan Arnborg, Derek G. Corneil and Andrzej Proskurowski. *Complexity of Finding Embeddings in a k-Tree.* SIAM Journal on Algebraic and Discrete Methods, 8(2):277-284, 1987.

[7] Marc Ayel. *Détection d'incohérences dans les bases de connaissances : SACCO.* Thèse d'état, Université de Savoie, 1987.

[8] Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, 1974.

[9] Alain Beauvieux. *Contrôler la cohérence d'une base de connaissances.* Huitièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1988.

[10] Claude Berge. *Graphs and Hypergraphs.* North-Holland, 1973.

[11] Pauline M. Berry. *Satisfying Conflicting Objectives in Factory Scheduling.* Proceedings of the IEEE International Conference on Artificial Intelligence Applications, Santa Barbara, California, 1990.

[12] Maurice Bruynooghe. *Solving Combinatorial Search Problems by Intelligent Backtracking.* Information Processing Letters, 12(1):36-39, 1981.

[13] Peter Burke and Patrick Prosser. *A Distributed Asynchronous System for Predictive and Reactive Scheduling.* Technical Report, University of Strathclyde, 1989.

[14] Peter Burke. *Scheduling In Dynamic Environments.* PhD Thesis, Department of Computer Science, University of Strathclyde, 1989.

[15] Philippe Caloud. *Distributed Motion Planning and Motion Coordination for Multiple Robots.* Working Paper, Stanford University, 1990.

[16] Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape and Mark Yim. *Indoor Automation With Many Mobile Robots.* Proceedings of the IEEE International Workshop on Intelligent Robots and Systems, Tsuchiura, Japan, 1990.

[17] Jacques Carlier et Philippe Chrétienne. *Problèmes d'ordonnancement : Modélisation / Complexité / Algorithmes.* Masson, 1988.

[18] Michel Cayrol et Pierre Tayrac. *Exploitation de la méthode du consensus dans les ATMS : la résolution CAH-correcte.* Huitièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1988.

[19] Michel Cayrol and Pierre Tayrac. *CAT-Correct and CCT-Correct Resolution, CAT-Correct Resolution in the ATMS.* Working Paper, Université Paul Sabatier, 1988.

[20] Michel Cayrol et Pierre Tayrac. *ARC : un ATMS basé sur la résolution CAT-correcte.* Revue d'intelligence artificielle, 3(3):19-39, 1989.

[21] David Chapman. *Planning for Conjunctive Goals.* Artificial Intelligence, 32(3):333-377, 1987.

[22] Po-Young Chu, Herbert Moskowitz and Richard T. Wong. *Robust Interactive Decision-Analysis (RID): Concepts, Methodology and System Principles.* Proceedings of the Twenty-Second Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, 1989.

[23] Edward G. Coffman Jr. (editor). *Computer and Job-Shop Scheduling Theory.* John Wiley and Sons, 1976.

[24] Anne Collinot and Claude Le Pape. *Controlling Constraint Propagation.* Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy, 1987.

[25] Anne Collinot et Claude Le Pape. *Comparaison de plusieurs modes d'utilisation d'un système d'ordonnancement flexible.* Rapport technique, Laboratoires de Marcoussis, 1988.

[26] Anne Collinot, Claude Le Pape and Gérard Pinoteau. *SONIA: A Knowledge-Based Scheduling System.* International Journal for Artificial Intelligence in Engineering, 3(2):86-94, 1988.

[27] Anne Collinot. *Revising the BB1 Basic Control Loop to Control the Behavior of Knowledge Sources.* Proceedings of the Second AAAI Workshop on Blackboard Systems, AAAI, Saint Paul, Minnesota, 1988.

[28] Anne Collinot. *Le problème du contrôle dans un système flexible d'ordonnancement.* Thèse de l'Université Paris VI, 1988.

[29] Anne Collinot and Claude Le Pape. *Controlling the Behavior of Knowledge Sources within SONIA.* Proceedings of the Twenty-Second Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, 1989.

[30] Anne Collinot and Claude Le Pape. *Testing and Comparing Reactive Scheduling Strategies.* Proceedings of the AAAI-SIGMAN Workshop on Manufacturing Production Scheduling, IJCAI, Detroit, Michigan, 1989.

[31] Anne Collinot and Claude Le Pape. *Adapting the Behavior of a Job-Shop Scheduling System.* International Journal for Decision Support Systems (to appear).

[32] Marie-Odile Cordier. *Informations incomplètes et contraintes d'intégrité : le moteur d'inférences SHERLOCK.* Thèse d'état, Université Paris XI, 1986.

[33] Jean-Marc David et Jean-Paul Krivine. *Utilisation de prototypes dans un système expert de diagnostic : le projet DIVA.* Septièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1987.

[34] Rina Dechter (editor). *Proceedings of the AAAI Workshop on Constraint Processing.* IJCAI, Detroit, Michigan, 1989.

[35] Johan De Kleer. *Choices without Backtracking.* Proceedings of the Fourth National Conference on Artificial Intelligence, Austin, Texas, 1984.

[36] Johan De Kleer. *An Assumption-Based TMS. Extending the ATMS. Problem Solving with the ATMS.* Artificial Intelligence, 28(2):127-224, 1986.

[37] Claude Delobel et Michel Adiba. *Bases de données et systèmes relationnels.* Dunod, 1982.

[38] Hubert De Ponthaud et Michel Venet. *FAREX : système expert de planification de missions de transport.* Neuvièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1989.

[39] Yannick Descotte and Jean-Claude Latombe. *GARI: A Problem Solver that Plans How to Machine Mechanical Parts.* Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, 1981.

[40] Yannick Descotte. *Représentation et exploitation de connaissances "expertes" en génération de plans d'actions. Application à la conception automatique de gammes d'usinage.* Thèse de troisième cycle, Institut National Polytechnique de Grenoble, 1981.

[41] Yannick Descotte and Jean-Claude Latombe. *Making Compromises among Antagonist Constraints in a Planner.* Artificial Intelligence, 27(2):183-217, 1985.

[42] Jacques Erschler. *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement.* Thèse d'état, Université Paul Sabatier, 1976.

[43] Jacques Erschler, Pierre Lopez et Catherine Thuriot. *Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement.* Working Paper, Université Paul Sabatier, 1989.

[44] Patrick Esquirol. *Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de contraintes.* Thèse de l'Université Paul Sabatier, 1987.

[45] Ronald Fagin and Joseph Y. Halpern. *Uncertainty, Belief, and Probability.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[46] Franck R. Field, Richard de Neufville and Joel P. Clark. *Fonction d'utilité : application au choix des matériaux.* Les cahiers du collège des ingénieurs, (1):63-73, 1988.

[47] Richard E. Fikes, Peter E. Hart and Nils J. Nilsson. *Learning and Executing Generalized Robot Plans.* Artificial Intelligence, 3(4):251-288, 1972.

[48] Barry R. Fox and Karl G. Kempf. *Opportunistic Scheduling for Robotic Assembly.* Proceedings of the IEEE International Conference on Robotics and Automation, Saint Louis, Missouri, 1985.

[49] Robert E. Fox. *MRP, KANBAN, or OPT. What's Best?* Inventories and Production Magazine, 2(4), 1982.

[50] Hervé Gallaire, Jack Minker and Jean-Marie Nicolas. *Logic and Databases: A Deductive Approach.* Computing Surveys, 16(2):153-185, 1984.

[51] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[52] Michael P. Georgeff. *Communication and Interaction in Multi-Agent Planning.* Proceedings of the Third National Conference on Artificial Intelligence, Washington, District of Columbia, 1983.

[53] Michael P. Georgeff and Amy L. Lansky. *Reactive Reasoning and Planning.* Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, Washington, 1987.

[54] Michael P. Georgeff and François Félix Ingrand. *Decision-Making in an Embedded Reasoning System.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[55] Malik Ghallab and Amine Mounir Alaoui. *Managing Efficiently Temporal Relations Through Indexed Spanning Trees.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[56] Malik Ghallab et Amine Mounir Alaoui. *Relations temporelles symboliques : représentations et algorithmes.* Revue d'intelligence artificielle, 3(3):67-115, 1989.

[57] Allen Ginsberg. *Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy.* Proceedings of the Seventh National Conference on Artificial Intelligence, Saint Paul, Minnesota, 1988.

[58] Eliyahu M. Goldratt and Jeff Cox. *The Goal. A Process of Ongoing Improvement.* North River Press, 1986.

[59] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[60] Michel Gondran and Michel Minoux. *Graphs and Algorithms.* John Wiley and Sons, 1984.

[61] Hans Werner Güsgen. *CONSAT: A System for Constraint Satisfaction.* Pitman, 1989.

[62] Joseph Y. Halpern. *An Analysis of First-Order Logics of Probability.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[63] Barbara Hayes-Roth and Frederick Hayes-Roth. *A Cognitive Model of Planning.* Cognitive Science, 3(4):275-310, 1979.

[64] James Hendler, Austin Tate and Mark Drummond. *AI Planning: Systems and Techniques.* AI Magazine, 11(2):61-77, 1990.

[65] William P.-C. Ho. *A Meta-Planning Model for Diminishing Resource Problems.* International Journal for Artificial Intelligence in Engineering, 3(2):114-120, 1988.

[66] Eric J. Horvitz. *Reasoning About Beliefs and Actions Under Computational Resource Constraints.* KSL Technical Report, Stanford University, 1987.

[67] Eric J. Horvitz, John S. Breese and Max Henrion. *Decision Theory in Expert Systems and Artificial Intelligence.* International Journal of Approximate Reasoning, 2(3):247-302, 1988.

[68] James R. Jackson. *An Extension of Johnson's Results on Job Lot Scheduling.* Naval Research Logistics Quarterly, 3(3):201-203, 1956.

[69] Joxan Jaffar and Jean-Louis Lassez. *Constraint Logic Programming.* Proceedings of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, West Germany, 1987.

[70] S. M. Johnson. *Optimal Two- and Three-Stage Production Schedules with Setup Times Included.* Naval Research Logistics Quarterly, 1(1):61-68, 1954.

[71] Arnold Kaufmann and Robert Faure. *Introduction to Operations Research.* Academic Press, 1968.

[72] Kevin M. Kelly, Louis I. Steinberg and Timothy M. Weinrich. *Constraint Propagation in Design: Reducing the Cost.* Working Paper, Rutgers University, 1988.

[73] Karl G. Kempf. *Manufacturing Planning and Scheduling: Where We Are and Where We Need To Be.* Proceedings of the IEEE International Conference on Artificial Intelligence Applications, Miami, Florida, 1989.

[74] Karl G. Kempf, Claude Le Pape, Stephen F. Smith and Barry R. Fox. *Issues in the Design of AI-Based Schedulers: A Workshop Report.* AI Magazine (to appear).

[75] Naiping Keng and David Y. Y. Yun. *A Planning/Scheduling Methodology for the Constrained Resource Problem.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[76] R. M. Kerr and R. N. Walker. *A Job-Shop Scheduling System Based on Fuzzy Arithmetic.* Proceedings of the Third International Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston, South Carolina, 1989.

[77] Stephen C. Kleene. *Mathematical Logic.* John Wiley and Sons, 1967.

[78] Robert Kowalski. *Algorithm = Logic + Control.* Communications of the ACM, 22(7):424-436, 1979.

[79] Peter B. Ladkin and Roger D. Maddux. *On Binary Constraint Networks.* Technical Report, Kestrel Institute, 1988.

[80] Anne Lalo. *TIBRE : un système expert qui teste les incohérences dans les bases de règles.* Huitièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1988.

[81] Jean-Claude Latombe. *Une application de l'intelligence artificielle à la conception assistée par ordinateur (TROPIC).* Thèse d'état, Institut National Polytechnique de Grenoble, 1977.

[82] Jean-Claude Latombe. *Failure Processing in a System for Designing Complex Assemblies.* Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.

[83] Hervé Le Lous et Vidal Cohen. *Théorie et pratique de la décision ; débat sur l'analyse du risque.* Les cahiers du collège des ingénieurs, (1):21-49, 1988.

[84] J. K. Lenstra and A. H. G. Rinnooy Kan. *Computational Complexity of Discrete Optimization Problems.* Annals of Discrete Mathematics, 4(1):121-140, 1979.

[85] Claude Le Pape. *SOJA: A Daily Workshop Scheduling System. SOJA's System and Inference Engine.* Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems, Warwick, United Kingdom, 1985.

[86] Claude Le Pape and Stephen F. Smith. *Management of Temporal Constraints for Factory Scheduling.* Proceedings of the Working Conference on Temporal Aspects in Information Systems, Sophia-Antipolis, France, 1987.

[87] Claude Le Pape. *Des systèmes d'ordonnancement flexibles et opportunistes.* Thèse de l'Université Paris XI, 1988.

[88] Claude Le Pape. *Représentation d'éléments de raisonnement en Géologie Appliquée : points de vue d'un informaticien.* Rapport interne, Elf Aquitaine, 1989.

[89] Claude Le Pape. *The Completeness of a Solution Maintenance Component.* Working Paper, Stanford University, 1989.

[90] Claude Le Pape. *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling.* Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, 1990.

[91] Claude Le Pape. *Simulating Actions of Autonomous Agents.* CIFE Technical Report, Stanford University, 1990.

[92] Claude Le Pape. *Intelligence artificielle et ordonnancement : une introduction.* Bulletin de l'association française pour l'intelligence artificielle, (4):11-12, 1990.

[93] Nadine Lerat and Witold Lipski Jr. *Nonapplicable Nulls.* Theoretical Computer Science, 46(1):67-82, 1986.

[94] Nadine Lerat. *Représentation et traitement des valeurs nulles dans les bases de données.* Thèse de troisième cycle, Université Paris XI, 1986.

[95] Bing Liu. *Scheduling via Reinforcement.* International Journal for Artificial Intelligence in Engineering, 3(2):76-85, 1988.

[96] Beatriz Lopez, Pedro Meseguer and Enric Plaza. *Knowledge-Based Systems Validation: A State of the Art.* AI Communications, 3(2):58-72, 1990.

[97] Eric Lumer and Bernardo A. Huberman. *Dynamics of Resource Allocation in Distributed Systems.* Technical Report, XEROX Palo Alto Research Center, 1990.

[98] Bernard Meltzer. *Prolegomena to a Theory of Efficiency of Proof Procedures.* Artificial Intelligence and Heuristic Programming, American Elsevier, 1971.

[99] David H. Mott, Jon Cunningham, Gerry Kelleher and Julie A. Gadsden. *Constraint-Based Reasoning for Generating Naval Flying Programmes.* Expert Systems, 5(3):226-246, 1988.

[100] A. P. Muhlemann, A. G. Lockett and C. K. Farn. *Job-Shop Scheduling Heuristics and Frequency of Scheduling.* International Journal of Production Research, 20(2):227-241, 1982.

[101] Tadao Murata. *Petri Nets: Properties, Analysis and Applications.* IEEE Proceedings, 77(4):541-580, 1989.

[102] Nicola Muscettola. *Planning the Behavior of Dynamical Systems.* Technical Report, Carnegie-Mellon University, 1990.

[103] Tin A. Nguyen, Walton A. Perkins, Thomas J. Laffey and Deanne Pecora. *Knowledge Base Verification.* AI Magazine, 8(2):69-75, 1987.

[104] H. Penny Nii. *Blackboard Systems Part Two: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective.* AI Magazine, 7(3):82-106, 1986.

[105] Nils J. Nilsson. *Principles of Artificial Intelligence.* Springer-Verlag, 1982.

[106] Peng Si Ow and Stephen F. Smith. *Two Design Principles for Knowledge-Based Systems.* Decision Sciences, 18(3):430-447, 1987.

[107] Peng Si Ow, Stephen F. Smith and Alfred Thiriez. *Reactive Plan Revision.* Proceedings of the Seventh National Conference on Artificial Intelligence, Saint Paul, Minnesota, 1988.

[108] Peng Si Ow and Thomas E. Morton. *The Single Machine Early/Tardy Problem.* Management Science, 35(2):177-191, 1989.

[109] H. Van Dyke Parunak. *Manufacturing Experience With the Contract Net.* Proceedings of the Fifth Workshop on Distributed Artificial Intelligence, Sea Ranch, California, 1985.

[110] Eric Pipard. *Détection d'incohérences et d'incomplétudes dans les bases de règles : le système INDE.* Huitièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1988.

[111] Jacques Pitrat. *Utilisation des connaissances déclaratives.* Publication 56 du Groupe de Recherche 22, Université Paris VI, 1985.

[112] Peter J. G. Ramadge and W. Murray Wonham. *The Control of Discrete Event Systems.* IEEE Proceedings, 77(1):81-98, 1989.

[113] Jean-François Rit. *Propagating Temporal Constraints for Scheduling.* Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, Pennsylvania, 1986.

[114] Jean-François Rit. *Modélisation et propagation de contraintes temporelles pour la planification.* Thèse de l'Institut National Polytechnique de Grenoble, 1988.

[115] J. A. Robinson. *A Machine-Oriented Logic Based on the Resolution Principle.* Journal of the ACM, 12(1):23-41, 1965.

[116] Marie-Christine Rousset. *Sur la validité des bases de connaissances : le système COVADIS.* Septièmes journées internationales sur les systèmes experts et leurs applications, Avignon, France, 1987.

[117] Marie-Christine Rousset. *Sur la cohérence et la validation des bases de connaissances : le système COVADIS.* Thèse d'état, Université Paris XI, 1988.

[118] David M. Russinoff. *PROTEUS: A Frame-Based Nonmonotonic Inference System.* Technical Report, Microelectronics and Computer Technology Corporation, 1987.

[119] Eric Rutten and Lionel Marcé. *Temporal Logics Meet Telerobotics.* Proceedings of the NASA Conference on Space Telerobotics, Pasadena, California, 1989.

[120] Earl Sacerdoti. *Problem Solving Tactics.* Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.

[121] Norman Sadeh and Mark S. Fox. *Preference Propagation in Temporal/Capacity Constraint Graphs.* Technical Report, Carnegie-Mellon University, 1989.

[122] Norman Sadeh and Mark S. Fox. *Focus of Attention in an Activity-Based Scheduler.* Proceedings of the NASA Conference on Space Telerobotics, Pasadena, California, 1989.

[123] Shimon Schocken. *A Framework for Comparative Analysis of Belief Revision Models in Rule-Based Systems.* Proceedings of the Twenty-Second Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, 1989.

[124] Raimund Seidel. *A New Method for Solving Constraint Satisfaction Problems.* Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, 1981.

[125] Prakash P. Shenoy and Glenn R. Shafer. *Axioms for Discrete Optimization Using Local Computation.* Working Paper, University of Kansas, 1988.

[126] Prakash P. Shenoy and Glenn R. Shafer. *Constraint Propagation.* Working Paper, University of Kansas, 1988.

[127] Prakash P. Shenoy and Glenn R. Shafer. *Axioms for Probability and Belief-Function Propagation.* Working Paper, University of Kansas, 1988.

[128] Herbert Simon. *Search and Reasoning in Problem Solving.* Artificial Intelligence, 21(1):7-29, 1983.

[129] Reid G. Smith. *The Contract Net: A Formalism for the Control of Distributed Problem Solving.* Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, 1977.

[130] Stephen F. Smith, Mark S. Fox and Peng Si Ow. *Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.* AI Magazine, 7(4):45-61, 1986.

[131] Stephen F. Smith. *A Constraint-Based Framework for Reactive Management of Factory Schedules.* Proceedings of the First International Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston, South Carolina, 1987.

[132] Stephen F. Smith and Juha E. Hynynen. *Integrated Decentralization of Production Management: An Approach for Factory Scheduling.* Proceedings of the ASME Annual Winter Conference, Boston, Massachusetts, 1987.

[133] Stephen F. Smith, Naiping Keng and Karl G. Kempf. *Exploiting Local Flexibility During Execution of Pre-Computed Schedules.* Technical Report, Carnegie-Mellon University, 1990.

[134] Richard M. Stallman and Gerald J. Sussman. *Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis.* Artificial Intelligence, 9(2):135-196, 1977.

[135] John A. Stankovic. *Misconceptions About Real-Time Computing. A Serious Problem for Next-Generation Systems.* IEEE Computer, 21(10):10-19, 1988.

[136] Mitchell S. Steffen. *A Survey of Artificial Intelligence-Based Scheduling Systems.* Proceedings of the Fall Industrial Engineering Conference, Boston, Massachusetts, 1986.

[137] Mark Stefik. *Planning with Constraints.* PhD Thesis, Department of Computer Science, Stanford University, 1980.

[138] Mark Stefik. *Planning with Constraints (MOLGEN: Part 1). Planning and Meta-Planning (MOLGEN: Part 2).* Artificial Intelligence, 16(2):111-169, 1981.

[139] Mark Stefik, Jan Aikins, Robert Balzer, John Benoit, Lawrence Birnbaum, Frederick Hayes-Roth and Earl Sacerdoti. *The Organization of Expert Systems, A Tutorial.* Artificial Intelligence, 18(2):135-173, 1982.

[140] Louis I. Steinberg. *Design as Refinement Plus Constraint Propagation: The VEXED Experience.* Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, Washington, 1987.

[141] Louis I. Steinberg. *Design = Top Down Refinement Plus Constraint Propagation Plus What?* Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Fairfax, Virginia, 1987.

[142] Katia Sycara, Stephen Roth, Norman Sadeh and Mark S. Fox. *An Investigation into Distributed Constraint-Directed Factory Scheduling.* Proceedings of the IEEE International Conference on Artificial Intelligence Applications, Santa Barbara, California, 1990.

[143] Austin Tate. *Planning in Expert Systems.* Proceedings of the Alvey IKBS Expert Systems Research Theme Workshop, Abingdon, United Kingdom, 1984.

[144] Austin Tate. *A Review of Knowledge-Based Planning Techniques.* Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems, Warwick, United Kingdom, 1985.

[145] Jean-Patrick Tsang. *Planification par combinaison de plans. Application à la génération de gammes d'usinage.* Thèse de l'Institut National Polytechnique de Grenoble, 1987.

[146] Peter Van Beek. *Approximation Algorithms for Temporal Reasoning.* Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, 1989.

[147] Peter Van Beek. *Reasoning about Qualitative Temporal Information.* Proceedings of the Eighth National Conference on Artificial Intelligence, Boston, Massachusetts, 1990.

[148] Pascal Van Hentenryck and Mehmet Dincbas. *Domains in Logic Programming.* Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, Pennsylvania, 1986.

[149] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[150] Marc Vilain and Henry Kautz. *Constraint Propagation Algorithms for Temporal Reasoning.* Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, Pennsylvania, 1986.

[151] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1953.

[152] Robert Wilensky. *Meta-planning.* Proceedings of the First National Conference on Artificial Intelligence, Stanford, California, 1980.

[153] David E. Wilkins. *Domain-independent Planning: Representation and Plan Generation.* Artificial Intelligence, 22(3):269-301, 1984.

[154] M. Yamamoto and S. Y. Nof. *Scheduling/Re-Scheduling in the Manufacturing Operating System Environment.* International Journal of Production Research, 23(4):705-722, 1985.