Constraints and AI Planning

Alexander Nareyek, Robert Fourer, Eugene C. Freuder,
Enrico Giunchiglia, Robert P. Goldman, Henry Kautz,
Jussi Rintanen and Austin Tate

Abstract

Tackling real-world problems often requires to take various types of constraints into account. Such constraint types range from simple numerical comparators to complex resources. This article describes how planning techniques can be integrated with general constraint-solving frameworks, like SAT, IP and CP. In many cases, the complete planning problem can be cast in these frameworks.

1 Introduction

The purpose of this article is to explore the various aspects of the interplay of constraints and planning, compare and highlight the differences between constraint programming (CP), integer programming (IP) and propositional satisfiability (SAT) in this context, and point out future directions[1].

The basic planning problem is usually given by an initial world description, a partial description of the goal world, and a set of actions/operators that map a partial world description to another partial world description. A solution is a sequence of actions leading from the initial world description to the goal world description and is called a plan. The problem can be enriched by including further aspects, like temporal or uncertainty issues, or by requiring the optimization of certain properties. In general, it is assumed that the reader is familiar with basic planning techniques and terminology (see, e.g., [Allen90] for basic literature).

[1]Note that we will not cover specialized reasoning schemes, like the management of temporal constraint networks (e.g., see [Dechter91]).

The management of constraints is an integral part of today's planning systems. Not only complex problem features, like scheduling and reasoning about numerical resources, can easily be expressed by a constraint-based framework. Simple orderings in partial-order planning systems as well as the exclusion relations of Graphplan-based planners can also be interpreted in this context. Recently, even planning systems in which planning problems are fully cast as a constraint satisfaction problem (CSP) have been developed.

Using constraints and related techniques as an underlying framework for problem-solving tasks – such as planning – has proven successful for many domains. The basic modeling units are *constraints* and *variables*. A constraint is an entity that restricts the values of variables. In order to use efficient solving techniques, most search frameworks use only a restricted scenario. In propositional satisfiability, constraints are restricted to propositional formulas, which constrain variables to a Boolean domain. In integer linear programming, linear inequalities can be applied to restrict numerical variables. Finally, constraint programming is the most general framework with no restriction on the types of constraints, although usually only variables with finite domains are considered. Such search frameworks were increasingly studied during the last several years and many successful application areas suggest an application to planning. Sections 2 to 4 describe and compare the different search frameworks and give examples how the frameworks can be applied for planning. Special emphasis is placed on constraint programming. The further development of this area and ideas for future research are discussed in Section 5.

2 Planning as Propositional Satisfiability

The problem of propositional satisfiability (SAT) is to decide whether a propositional formula is satisfiable or not. A propositional formula consists of two-valued variables (`true` and `false`) that are related via the operators $\neg$ ("not"), $\vee$ ("or" or disjunction) and $\wedge$ ("and" or conjunction). Most solvers require that the problem be stated as a conjunctive normal form (a conjunction of disjunctions).

SAT-solving techniques include refinement approaches, which are mostly based on the Davis-Putnam procedure [Davis60], like Satz-Rand [Gomes98] and Chaff [Moskewicz01], and local-search methods like GSAT [Selman92] and SDF [Schuurmans00].

Planning as Satisfiability is a paradigm originally proposed by Kautz and Selman [Kautz92]. The idea is simple. Suppose that we have a finite description of the planning problem; to be specific, we assume a STRIPS description D with finitely many objects [Fikes71]. Since the domain is finite, it is possible to write a propositional formula TR_i whose models are one-to-one with the possible transitions in D. There are several possible ways to write such TR_i starting from a STRIPS description (see, e.g., [Kautz96]). For our goals, it suffices to say that in TR_i, there is a propositional variable A_i for each ground action A, and two propositional variables F_i and F_{i+1} for each ground fluent[2] F in D. Intuitively, F_i represents the value of F at time i, and similarly for A_i and F_{i+1}.

Then, if D is deterministic and we are given a single initial state represented with a formula I, a goal state represented with a formula G is reachable in n steps if the formula

$$I_0 \wedge (\bigwedge_{0 \leq i < n} TR_i) \wedge G_n \tag{1}$$

is satisfiable. In the above formula, I_0 is the formula obtained from I by replacing each state variable F with F_0, and G_n is obtained from G by replacing each state variable F with F_n.

This simple idea has had a lot of impact in the planning community, mainly because it has led to very impressive results (see [Kautz96]). Still, it is clear that for very large values of n (say, exponential in the size of D), the size of (1) becomes problematic, and "planning as satisfiability" may no longer be feasible. However, in many cases we look for short plans, and in these cases planning as satisfiability can be a winning approach. Indeed, similar considerations and results have been obtained also in the area of formal verification. In this setting, TR_i encodes the transition relation of the system under analysis, $\neg G$ is the property we want to verify, and I represents the set of possible initial states: it is possible to violate the property in the first n ticks if (1) is satisfiable, and systems based on this idea are far more effective than the others if n is small (see, e.g., [Biere99]).

Given the above, it is clear that the planning as satisfiability approach is not restricted to work with STRIPS domain descriptions. We may start with a domain description written in any language for specifying domains, and as long as

[2] The term *fluent* is equivalent to a state variable.

- the domain is deterministic and there is a single initial state, and
- we have a propositional formula TR_i encoding all the possible transitions,

we can determine plans of length n by satisfying Formula (1).

3 Operations Research Approaches to AI Planning

The concept of an operations research (or OR) approach has long been influential in AI planning. Much of the early work on heuristic search was inspired by OR methods. For example, the Nonlin hierarchical partial-order planner [Tate77] was developed within a project entitled "Planning: A Joint AI/OR Approach" and incorporated OR methods for temporal constraint satisfaction. The ZENO temporal planner [Penberthy94] adopted OR data structures and algorithms for handling linear inequality constraints. Indeed, if "operations research" is interpreted broadly as the study of operations, then it can encompass almost any AI planning method.

Nevertheless, for highly combinatorial problems such as AI planning, there exists a distinctive *integer programming* (IP) approach that is arguably the one most widely associated with OR. This approach involves first casting the problem as the minimization or maximization of a linear function of integer-valued decision variables, subject to linear equality and inequality constraints in the variables. A solution is then sought through a general-purpose *branch-and-bound* procedure based on the idea of a search tree [Wolsey98]. The branching of the tree corresponds to the division of the original problem into progressively more constrained sub-problems, with leaf nodes representing individual solutions. A full tree search would thus involve a complete enumeration, but in practice an optimum is often determined after examining only a minuscule fraction of the nodes — through the use of bounds derived from relaxations of the node sub-problems, in conjunction with a variety of ingenious heuristic procedures.

For classical AI planning problems, no minimization or maximization is required, and the branch-and-bound procedure may instead be used simply to seek a feasible solution to the constraints. Still, the search for a solution may be guided by constructing an objective such as minimization of the number of actions taken.

Vossen *et al.* [Vossen01] report experiments in AI planning that are representative of experience in combinatorial optimization via the integer programming approach. They consider first a straightforward formulation, based on the SAT representation employed by Graphplan [Blum97], in terms of integer variables that take the value 1 if a certain condition holds and 0 otherwise:

`Do[a,t]` $= 1$ iff action `a` is taken in timestep `t`
`True[f,t]` $= 1$ iff fluent (assertion) `f` is true in timestep `t`

Linear constraints on these variables insure that state transitions are consistent with actions. Because the variables take zero-one values rather than the true-false values of a SAT problem (Section 2), the IP's operators are arithmetic rather than logical; but the assertions expressed by the IP's constraints correspond directly to the logical conditions in the SAT representation.

As a concrete example, consider a case of a "blocks world" having an arm to pick up, hold or put down at least two blocks, A and B. There must be a constraint to implement the rule that the arm may not put down block B at timestep 3 unless it holds that block at that time:

```
Do['put_down_B',3] ≤ True['hold_B',3]
```

Another kind of constraint ensures that the arm may not hold block B at timestep 3 unless one of the three actions compatible with holding it was performed at timestep 2:

```
True['hold_B',3] ≤
   Do['pick_up_B',2] + Do['unstack_B_A',2] + Do['noop_hold_B',2]
```

Further constraints rule out the performance of incompatible actions in the same timestep, such as both putting down and picking up block B:

```
Do['put_down_B',3] + Do['pick_up_B',3] ≤ 1
```

Finally, a few constraints specify the initial state and requirements for the final state. All of the necessary constraints of these types can be derived mechanically from the contents of a STRIPS representation of the problem.

Only the integrality of the `Do` variables needs to be enforced, as the `True` variables may simply be constrained to lie in the interval [0,1] and will be forced by the constraints to take only the values 0 and 1 at an optimal solution. Even so, as reported in [Vossen01], a highly respected branch-and-bound implementation was unable to prune the search tree sufficiently to avoid prohibitive increases in execution time and memory requirements for over half of the test problems considered.

As is often the case in integer programming, much better results may be produced by a "tighter" representation that is considerably less intuitive but which yields better bounds for pruning the search tree. In one such representation, also derived by [Vossen01], the `True` variables are replaced by four collections of variables similarly indexed over fluents and timesteps:

`Do[a,t]`	= 1 iff action `a` is taken in timestep `t`
`Keep[f,t]`	= 1 iff `f` is kept true in timestep `t`
`Add[f,t]`	= 1 iff `f` is not a precondition but is added in timestep `t`
`PreAdd[f,t]`	= 1 iff `f` is a precondition but is not deleted in timestep `t`
`PreDel[f,t]`	= 1 iff `f` is a precondition and is deleted in timestep `t`

For each fluent and timestep, three constraints define the relations between the new variables. Thus for example at most one of `Keep`, `Add`, `PreDel` and one of `Keep`, `PreAdd`, `PreDel` can apply to the fluent that represents holding block B at timestep 3:

```
Keep['hold_B',3] + Add['hold_B',3] + PreDel['hold_B',3] ≤ 1
Keep['hold_B',3] + PreAdd['hold_B',3] + PreDel['hold_B',3] ≤ 1
```

Moreover, any of `Keep`, `PreAdd`, `PreDel` can apply only if one of `Keep`, `PreAdd`, `Add` applied at the previous timestep:

```
Keep['hold_B',3] + PreAdd['hold_B',3] + PreDel['hold_B',3] ≤
Keep['hold_B',2] + PreAdd['hold_B',2] + Add['hold_B',2]
```

Additional constraints relate these new variables to the `Do` variables, which again are the only ones that need to be explicitly constrained to be integer. For instance, the constraints must insure that "holds block B" can become true at timestep 3 if and only if the arm either picks up or unstacks block B at timestep 3:

```
Add['hold_B',3] ≥ Do['pick_up_B',3]
Add['hold_B',3] ≥ Do['unstack_B_A',3]                              (2)
Add['hold_B',3] ≤ Do['pick_up_B',3] + Do['unstack_B_A',3]
```

The branch-and-bound solver performed much more strongly on such a formulation, solving all of the problems considered (though with computation times over 1000 seconds and search trees of over 80000 nodes for the two hardest cases). Even so, conventional test problems were solved even more efficiently by use of Blackbox [Kautz98], a package that combines more specialized processing of AI planning problems with propositional satisfiability solvers.

The effectiveness of general-purpose IP software has improved considerably, however, in the few years since the tests described above were run. Integer programming may also be more suitable for temporal or resource-oriented planning problems, because these involve larger numerical variable domains that are hard to encode using propositional satisfiability approaches. Another alternative is to combine solvers of both types for those tasks, a system being realized by Wolfman and Weld [Wolfman99], in which a SAT solver is used to solve the core planning problem and IP is used to take care of the numerical relations.

The flexibility of integer programming has been further strengthened by the development of several *modeling languages* [Bisschop82, Fourer83, Kuip93] that permit general formulations to be written in a way that is concise and natural for modelers, yet is capable of efficient processing by computer. For example the collection of *all* constraints of the form (2), enforcing the relationship between the `Add` and the `Do` variables for all combinations of fluents and timesteps, can be written in the AMPL language [Fourer03] as

```
subj to DefnAddOnlyIf
   {f in FLU, a in ADD[f] diff PRE[f], t in 1..T}:
      Add[f,t] >= Do[a,t];

subj to DefnAddIf {f in FLU, t in 1..T}:
   Add[f,t] <= sum {a in ADD[f] diff PRE[f]} Do[a,t];
```

This is a direct transcription of mathematical statements that appear in a conventional formulation of the integer program. The `sum` operator denotes the mathematical Σ, for example, character pairs `>=` and `<=` represent the inequality relations $\geq$ and $\leq$, and expressions of the form `{...}` stand for indexing sets. By providing high-level, symbolic expressions for indexed collections of constraints, modeling languages and their supporting environments do much to encourage experimentation with a variety of formulations.

4 Applying Constraint Programming to Planning

The constraint programming approach is characterized by more natural or convenient formulations than those described in the previous sections. Problems are described as so-called *constraint satisfaction problems (CSPs).* A CSP consists of

- a set of variables $x = \{x_1, \ldots, x_n\}$
- where each variable is associated with a domain $d_1, ..., d_n$
- and a set of constraints $c = \{c_1, ..., c_m\}$ over these variables.

The domains can be symbols as well as numbers, continuous or discrete (e.g., “block A”, “13”, “6.5”). Constraints are relations between variables (e.g., “x_a is on top of x_b”, “$x_a < x_b \times x_c$”) that restrict the possible value assignments. Constraint satisfaction is the search for a variable assignment that satisfies the given constraints. Constraint optimization requires an additional function that assigns a quality value to a solution and tries to find a solution that maximizes this value.

The CSP structure of variables and constraints can be represented as a graph with variables and constraints as nodes. A variable node is linked by an edge to a constraint node if the corresponding constraint relation includes the variable. We will often refer to this *constraint graph* instead of a CSP in the following.

Advantages of the constraint programming framework become apparent in slightly more complex settings, e.g., when incorporating resources and time. As an example scenario for the following comparison, a number of tasks with specific durations share a resource and their overlap is thus to be prevented.

The mathematical framework of integer linear programming and the propositional formulas of SAT are highly restricted, and a problem must be translated into a very specific form, which can often only be done by experts. For the SAT approach, representing the numerical aspects of the task durations of the non-overlap example above is already hardly possible because the propositional SAT variables can represent only qualitative differences – unlike CP and IP which can also represent quantitative information. Of course, discrete numbers can also be modeled in SAT by using a propositional variable for each value of the discrete domain. But this is a highly unsuitable approach, not only because of its immense costs but also because this destroys the information about the values' ordering relation, which should be exploited during search.

For an IP approach, the inherent disjunction in the non-overlap problem (task A may preceed task B *or* task B may preceed task A) blows up the size of a representation by inequalities enormously. Constraint programming, by contrast, enables to use higher-level (so called "global") constraints. A formulation could simple look like `nonoverlap(taskA_begin, taskA_duration, taskB_begin, taskB_duration)`. Add-on languages like the already mentioned AMPL for IP relax this statement only slightly because many problem aspects cannot be adequately translated to the linear inequality context.

Global constraints do not only simplify modeling. The IP and SAT approach will break the problems down into their specific frameworks and may then scan the resulting specifications for structures on which to apply specific solution strategies. But – although many efficient methods have been developed – the propositional clauses of SAT and the linear inequalities of IP are scarcely able to exploit the higher-level domain knowledge anymore to support search (see also [Milano00]). This is not the case for constraint programming, whose high-level constraints are able to capture domain-specific dependencies. The availability of such global constraints and corresponding solution techniques has been a very important factor in the success of constraint programming.

Another advantage of using CP for planning is that there are lots of planning decisions with discrete alternatives, and the performance of OR methods declines sharply as the number of integer variables increases. However, there are ongoing efforts to combine constraint programming with integer programming [Hooker00, Milano00, Refalo99].

We adopt a model-based point of view here, defining constraint-based planners by the way the planning problem is cast — using explicit constraints. The types of these constraints are, as discussed before, not bound to propositional clauses or linear inequalities. By contrast, a widely used solution method for constraint satisfaction problems — propagation techniques — are sometimes seen as the essential ingredient of constraint programming, and planners applying this technique in one form or another are called constraint-based. From this point of view, however, nearly all planning systems would fall into this category because even the procedure of conventional total-order planners can be interpreted as excluding infeasible refinements by "propagating" state information.

The very first appearance of a system involving constraints was the MOLGEN planner [Stefik81]. More recent planners based on this idea include Descartes [Joslin96], *parc*PLAN [Lever94, Liatsos00], CPlan [Beek99], the approach of Rintanen and Jungholt [Rintanen99], GP-CSP [Do00], the approach of Jacopin and Penon [Jacopin00], MACBeth [Goldman00], the Excalibur agent's planning system [Nareyek01a] and EUROPA [FrankTA]. They can be grouped into categories, which are described in the following subsections:

- **Planning with Constraint Posting:**
  - CP is used for subproblems of planning
  - limited interaction between CSP solving and the actual planning process
- **Planning with Maximal Graphs:**
  - a large CSP is constructed, involving all possible planning options up to a specific plan size
  - full interaction between value- and structure-based problem aspects
  - does not scale well
- **Completely Capturing Planning within Constraint Programming:**
  - expresses the complete planning problem with the CP framework
  - requires an extended CP framework to cover different possible graph structures
  - full interaction between value- and structure-based problem aspects

However, we start with a section describing popular search techniques applied in constraint programming.

4.1 Search Techniques

Different search techniques can be employed for solution extraction – most common is a tree-based refinement search. In contrast to the branch-and-bound search mechanisms of Section 3, no continuous relaxations are applied. Instead, a variety of branch selection, domain tightening, and simplification procedures are applied at all nodes to reduce the extent of the tree that must be searched.

The first central component are *variable/value selection* techniques. A variable is selected for being assigned with a specific value and then the value itself is determined. This assignment process is called *labeling* and is iterated until all variables are assigned with values. Numerous variable and value ordering heuristics have been proposed, such as smallest-domain-first or smallest-value-first [Sadeh96]. In the case of an inconsistency (i.e., a constraint is violated), backtracking is triggered [Kondrak97].

The second important component is so-called *propagation* or *consistency* techniques. After each labeling step, reductions of the other variables' domains can be deduced, which may in turn lead to further domain reductions. For example, we have two variables A and B with domains of $\{1...10\}$ and a constraint $B > A$. In the case of a labeling of A to 5, the propagation will entail a domain reduction of B to $\{6...10\}$. Generic propagation techniques vary in which detail consequences are considered, trading off deduction costs against search costs. Examples of propagation techniques are AC-7 [Bessière95] and PC-4 [Han88].

A special type of constraints are the already mentioned global constraints. These are higher-level constraints and usually provide functionality to perform highly specialized and very efficient propagations.

A different paradigm to search for solutions is that of *local search*. In local search, the domains of the variables are not stepwise reduced, but concrete assignments for the variables are changed in an iterative way. However, in respect to constraint programming, search techniques based on local search are not very popular so far. This may be due to the traditional logic-programming framework for constraint programming. Nevertheless, many new methods have been proposed, especially in the last years. Some examples are the min-conflicts heuristic [Minton92], GENET [Davenport94] and an approach based on global constraints [Nareyek01b].

4.2 Planning with Constraint Posting

This section describes an approach of utilizing constraint programming for planning by posting/adding (and retracting in case of backtracking) constraints during a conventional planning process. Constraint satisfaction is used here as an add-on for planning to check the satisfaction of restrictions such as numerical relations. For example, MOLGEN [Stefik81], MACBeth [Goldman00] and the planning procedure of Jacopin and Penon [Jacopin00] apply this scheme. A more integrated approach is implemented in the Descartes [Joslin96] and *parc*PLAN [Lever94, Liatsos00] systems, which use constraint postings not only for numerical values, but also for postponing some of the decisions of action choice. In a wider context, also systems like I-Plan/O-Plan [Currie91, Tate94, Tate00], IxTeT [Laborie95] and HSTS [Muscettola94] fall into this category. In the following, we give an example how constraint posting can be integrated in hierarchical task network planning.

Hierarchical Task Network (HTN) planning [Erol94a] (sometimes referred to as *decomposition planning*) often provides an alternative to the more conventional means/ends or "first principles" planning, in particular, when planning problems involve the application of standard operating procedures (or other well-established process fragments), rather than the "puzzle mode" planning that characterizes domains like the Blocks world. Besides plan generation, HTN systems also address the problem of modeling domains in which planning takes place, make it easy to understand, control and communicate tasks, plans, intentions and effects between agents [Paolucci00], and allow users and computer systems to cooperate and work together using a "mixed initiative" style [Goldman00, Tate00]. Finally, HTNs provide a natural way to stipulate global constraints on plans, meshing well with the needs of systems that combine planning and constraint satisfaction [Erol94b].

Military small unit operations, emergency medical treatment, civilian aviation under free flight, and some team robotics applications fit this model. For lack of a better term, these problems are called "tactical planning" problems. While tactical planning problems involve application of previously-known procedures, that does not mean they are easy to solve. Typically, these problems include a rich and diverse set of constraints from the application domain which can strongly restrict the solution space. These constraints may be available only through ancillary "black box" problem solvers.

An HTN least commitment planning approach has been used for many years in practical planning systems such as NOAH [Sacerdoti75], Nonlin [Tate77] and SIPE [Wilkins88]. Two existing planners that involve a combination of HTN planning and constraint satisfaction techniques are AIAI's I-Plan/O-Plan and Honeywell Laboratories' MACBeth system. When HTN planning is joined with a strong underlying constraint-based ontology of plans it can provide a framework in which powerful problem solvers based on search and constraint reasoning methods can be employed and still retain human intelligibility of the overall planning process and the plan products that are created.

The main job of a tactical planner is to select and map appropriate procedures or processes onto different situations. In the process, the planner must explore and compare multiple options, and allow users to explore and compare multiple options. This is shown for O-Plan in figure 1. The planner must check ancillary constraints involved and where possible use the information to restrict the solution space. The human user must be able to intervene freely during the planning process, both in heuristic guidance ("I suggest to try at first to fly in formation during the ingress phase of the mission") and constraints ("I want you to fly over this point").

HTN planners differ from conventional first principles planners in the way they generate sub-goals. First principles planners generate sub-goals in order to satisfy unsatisfied preconditions of operators they would like to execute. For example, in blocks world, the desire to stack block A on block B might cause a first principles planner to generate a sub-goal to clear block B. Sub-goaling continues until all of the sub-goals have been satisfied by adding operators or by the initial conditions (in our example above, if block B is already clear in the starting state). On the other hand, an HTN planner decomposes complex goals into simpler and simpler goals, building a tree [3] from the initial complex goals down to leaves that are all executable primitives. At each level, additional constraints can be posted/added to refine the feasibility testing.

For example, an HTN planner for military air missions might start with a "perform reconnaissance" top level goal, whose parameters indicate what location is to be surveyed. Matching this goal would be a task network (or method) that would decompose the top-level goal into three sub-goals: "perform ingress," "overfly objective," and "perform egress." There would be additional constraints that indicated that the three sub-goals should be

[3] or graph, if operators may be used for more than one purpose.

achieved in the order listed, and that constrained the ingress phase to bring the aircraft to an appropriate staging location, the overflying phase to carry the aircraft from staging location over the target and the egress phase to carry the aircraft back to base from the target area. Additional constraints might be added for resource use, perhaps to ensure that the aircraft not exceed its fuel allowance, or complete its mission by a given deadline.

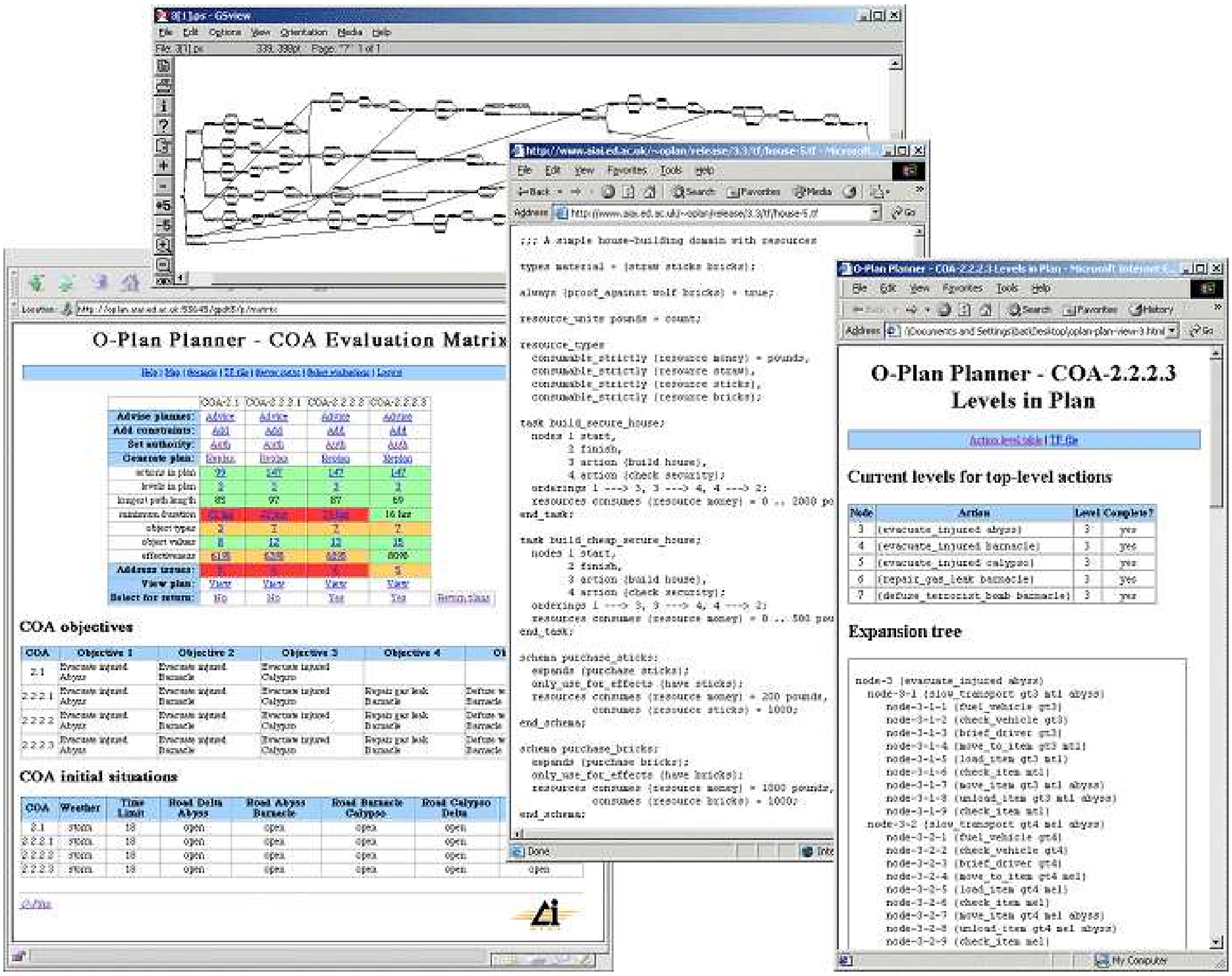


Figure 1: Interaction with the O-Plan system.

An advantage of HTN planning that makes it particularly suited to planning with constraints, is that HTN planning makes it convenient to specify constraints that cover sub-spaces of a plan. For example, it is trivial to create a plan for air travel that specifies that the airport used for departure and the airport used for return must be the same. This is non-trivial for a conventional STRIPS-style planner, particularly if the initial position of the agent and its desired ending points are not at the airport. Indeed, it is difficult to use a conventional STRIPS planner to create plans where an agent starts in an initial position (or state), achieves some goal, and then returns to the initial position (or state). Again, this is trivial in an HTN planner. In order to build such plans, a STRIPS planner would have to state-encode the constraint in some way, for example, adding a "takeoff_airport" fluent. This is feasible, of course, but imagine what happens if one wishes to take two flights, or four, or eight, possibly with multiple agents, and interleaved.

Related to the above point, it is simple for an HTN planner to represent resource use over sections of a plan. For example, a flight plan operator might have three sections of the plan, each of which would have an associated variable for its fuel use. Overall fuel use would be stipulated to be the sum of these variables, for example:

$$FuelUse = IngressFuelUse + OverDestFuelUse + EgressFuelUse$$

In turn, the operators for the three phases of the flight would provide further equations on fuel use. This would make for efficient constraint propagation even when planning occurs at arbitrary points in the chronological sequence. For example, if a reconnaissance aircraft were required to loiter over a particular Destination for a certain amount of time, the planner might be able to determine $OverDestFuelUse$ early in the planning process. The equations specified in the HTN would make it easy to propagate the effects of that constraint over the entire plan. In contrast, conventional STRIPS-style planners would find it difficult to reason about such resources except from plan prefixes or suffixes. The HTN planners expressive advantage could translate into greater search efficiency by allowing planning and constraint solving more freedom to apply most-constrained-first sorts of heuristics.

HTN planning is a promising domain for an application of constraint-posting techniques because the expressive power of HTNs make it easy to specify global constraints and make them available to constraint solvers.

4.3 Planning with Maximal Graphs

In contrast to the approach of the previous section – in which constraints were stepwise posted/added depending on the current plan refinement state – the techniques described in this section require that all constraints of the domain are posted at once.

Similar to SAT/IP approaches, a restricted planning problem is encoded as so-called "conditional" CSP[4]. The CSP is constructed such that it includes all possible plans up to a certain number of actions. Actions and related sub-parts of the CSP can be activated or deactivated. Activation and deactivation can be interpreted as SAT's TRUE and FALSE values for the ground actions/fluents (0 or 1 values in case of IP). More compact representations, like the one given below, are possible as well. We call approaches like this to be based on *maximal graphs/structures* because a representation that encompasses all possible options must be constructed.

Planning systems like CPlan [Beek99], the approach of Rintanen and Jungholt [Rintanen99], and GP-CSP [Do00] follow this line. The advantage of constructing a structure with all possibilities is that decisions can easily be propagated through the whole CSP, and future decision options that become infeasible in consequence can be eliminated. For planning problems, the optimal size (e.g., the number of actions) is not known in advance, so that a stepwise extension of the CSP structures must be performed if no solution can be found. Similar to SAT and IP formulations, which can also be classified as approaches based on maximal graphs, maximal structures can soon result in an unmanageble size of the CSP if a planning problem is slightly larger.

A constraint programming variation on the formulations of Section 3 could define fewer `Do` variables but with larger domains, taking members of the set of actions, rather than 0 and 1, as their values:

`Do[t]` = the action performed at timestep `t`

Also a number of logical conditions, such as set membership and equivalence, that would have to be translated to numerical inequalities for integer programming, can be expressed directly in a constraint programming model. For example, if we write `ADD[f]` and `PRE[f]` for the subsets of actions that add fluent `f` and that have fluent `f` as prerequisite, respectively, then the

[4] This type of CSP was formerly also called "dynamic" CSP [Mittal90]. What many people today understand as a dynamic CSP is however different, and we therefore use the term "conditional" here.

three integer programming constraints that relate the `Add` and `Do` variables for holding block B at timestep 3 can be subsumed by one more straightforward and general constraint:

```
Add['holding_B',3] <=>
    Do[3] in ADD['holding_B'] & Do[3] not in PRE['holding_B']
```

In many modeling languages for constraint programming, such as the OPL language [VanHentenryck99], these constraints can very conveniently be expressed. Consider again the relationship between the `Add` and `Do` variables; the entire collection of constraints that enforce this relationship, for all combinations of fluents and timesteps, can be written in OPL as

```
forall (f in FLU, t in 1..T)
   Add[f,t] <=> Do[t] in ADD[f] & Do[t] not in PRE[f];
```

As a note for actually solving the CSP specified above, for best results the search should focus on the `Do` variables, since they effectively define all of the others, but this information may have to be conveyed explicitly to the solver by someone familiar with the model.

4.4 Completely Capturing Planning within Constraint Programming

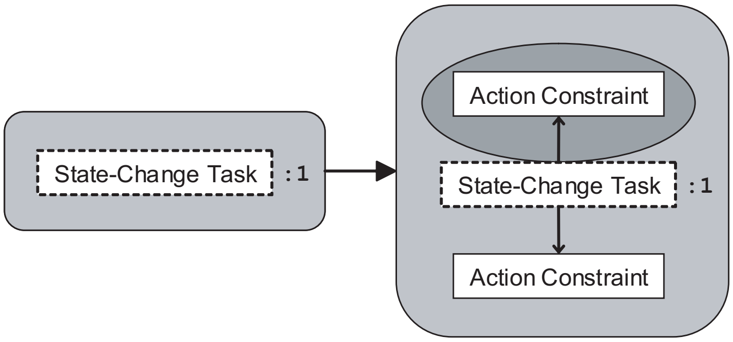
The approaches of using maximal graphs that restricts to a finite number of variables might be suitable for small problems that belong to the complexity class NP. As the complexity of planning problems increases, representation becomes difficult or impossible (same for the related SAT and IP approaches). Such computational limitations arise for example in simplest forms of planning with uncertainty and incomplete information, conformant and conditional planning. These are complete problems for the complexity class Σ_2^p, even when restricted to plans of polynomial size, and there does not appear to be natural ways of restricting these problems so as to force them to NP. For example, probabilistic planning with the same plan size restriction is complete for $\mathrm{NP}^{\mathrm{PP}}$. This means that CSP representaions for these problems cannot be generated in polynomial time.

For many simple cases, if the approach of maximal graphs is applicable, the sizes of the CSP graphs, even when they are polynomial, may not be practical because even a linear increase in the number of variables in the representation may translate to an exponential increase in the time needed in solving the problem. And most frustrating, the stepwise necessary graph-structure extension phases (if no plan was found for the current CSP structure) imply a given exploration path of the problem space thereby. This does not really fit the general idea of constraint programming – that a specification of a problem should not be mixed with search control, which makes CP solutions so easily adaptable and widely applicable.

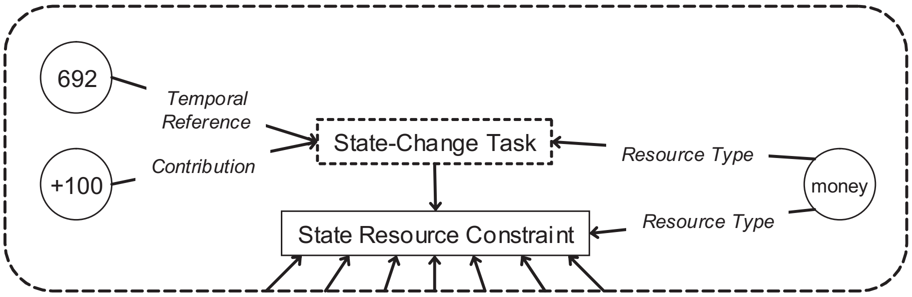
The core of the problem for approaches that try to completely capture planning within constraint programming is that the CP paradigm must somehow be extended such that the CSP graph can be changed dynamically instead be being built a priori. Section 4.2 presented an approach in which the CSP graph was built stepwise by an external planning process. However, only a subproblem of planning is covered within CP thereby, separating value- and structure-based problem aspects. In the approach discussed in the following [Nareyek01a], an extended CP framework is used to seemlessly integrate the satisfaction/optimization of both aspects.

In this approach, the structure of the CSP is not created in advance to cover all possible plans. So-called *structural constraints*, which represent restrictions on the possible constraint graphs, are defined and the search then includes the search for a correct constraint graph as part of the optimization process. The satisfaction and/or optimization of variables' values and graph structure can be seamlessly interleaved by the search process, and search can freely move around in the full planning problem's search space. In addition, the approach allows one to drop the closed-world assumption, i.e., it is not restricted to handle only a pre-defined number of objects in the world. The approach was developed for the Excalibur agent's planning system, which uses a local-search approach for solving the problems cast this way.

A structural constraint can for example test the relation that a state-change task, which is a collection of variables representing an event to change the state of a specific property, is always connected to exactly one action constraint. The role of the action constraint is to ensure that connected precondition/operation/state-change tasks together form a valid action. If this structural constraint holds for all state-change tasks that exist in a constraint graph (and potential other structural constraints hold as well), the graph is called *structurally consistent*. Figure 2 shows the structural constraint and a part of a constraint graph (variables are shown as circles, constraints as rectangles), which is structurally inconsistent because the state-change task of adding 100 units of money at time 692 is not part of an action, i.e., is not connected to an action constraint as required by the structural constraint.



An example of a structural constraint. The constraint applies at any part of the graph where the left side matches, and is fulfilled if the right side matches then as well (the dark area of the right side describes a "negative" match, i.e., exactly one action constraint is required to be connected to the state-change task).



A partial view of a structurally inconsistent constraint graph. The structural constraint above matches the state-change task but does not find a connected action constraint.

Figure 2: Extending the CP framework by structural constraints.

The inconsistency resolution (and optimization) of the local-search-based approach of the Excalibur agent's planning system does not only involve changing values of variables, but changes the constraint graph as well. For example, a state resource constraint, which projects a specific property's state over time and checks if related precondition tasks are fulfilled, may add a new state-change task to the constraint graph to resolve an unsatisfied precondition. The previously described structural constraint will be unsatisfied in consequence because the new state-change task is not connected to any action constraint (as in the figure above). Search might then in turn trigger the addition of a new action constraint in order to get back to structural consistency.

A similar approach in the direction of completely capturing planning within constraint programming has been presented by Frank and Jónsson [FrankTA]. It is more focused on specific interval elements than on general structural constraints. So-called *compatibilities* are defined, which represent implicative relations among interval elements. This enables to require the existence of further intervals and constraints in specific situations.

5 Future Directions

In the previous sections, we have tried to give the reader an overview of methods to apply constraint techniques to planning. Interest in the use of constraint techniques for AI planning problems has grown considerably in the recent years, and we believe that the area has a great potential for the future. One reason is that the use of more general modeling techniques for planning will promote an easy extendability toward additional problem aspects.

5.1 Feature Integration

The principal advantage, but also their principal disadvantage, lies in the generality of constraint-based methods. Many kinds of additional requirements can be introduced into the AI planning problem, e.g., in case of IP, by simply adding linear inequalities to an integer program and re-applying the general branch-and-bound solver. Comparable additions tend to necessitate more substantial changes to specialized planning systems.

An example of incorporating additional problem aspects is the area of integrating planning with resource and scheduling features. The complex numerical and logical relations involved make it hard to add such features in conventional planning systems. By contrast, many of the previously mentioned planning systems based on constraint handling already offer it. They often profit from the enormous body of scheduling and resource research that has already been done in the IP and CP communities, and the open nature of the solvers enables an easy integration. An overview of related solving techniques can be found in [Laborie01].

Another interesting direction is the synthesis of constraint-based and first-order planning technology. A large body of work was developed based on turning first-order logic into a tool for practical programming. GOLOG and its successors [Levesque97] pushed the expressive power of logic-based planning and control languages. While retaining the formal semantics of the situation calculus, GOLOG has come to include partial knowledge, sensing actions, conditionals, iteration, interrupts, and other constructs. Recently [Boutilier00] added the ability to handle quantitative probabilistic information and decision-theoretic control.

The greater expressive power of GOLOG comes at a computational cost: although GOLOG includes non-deterministic choice operators, they must be used with great care to avoid a combinatorial explosion. GOLOG's inference engine is simple Prolog-style chronological backtracking. In practice the GOLOG user needs to write a nearly-deterministic logic program. In short: GOLOG excels at representing open-ending domains and expressing explicit control knowledge about planning, while many constraint-based approaches excel at combinatorial search in large state-spaces.

A promising future research direction is therefore to develop a synthesis of constraint-based and first-order planning. One approach would build upon the insight that at a high enough level of abstraction many planning domains are essentially propositional. Algorithms based on Graphplan [Blum97] and satisfiability testing could be used to search very large state-spaces of candidate abstract plans. First-order logic programming and decision-theoretic techniques could then be used to expand the abstract plans to executable specifications.

An alternative (complementary) approach to synthesizing the two would build upon Ginsberg's suggestion that the role of "commonsense knowledge" is to transform a real-world problem instance into a small propositional core that can be solved by brute-force search [Ginsberg96]. The planner would use a rich set of first-order problem reformulation rules to identify relevant objects and operators and abstract away irrelevant parts of the problem statement. The instantiation of the relevant pieces creates the propositional core.

In either case, the goal is the same: to harness the raw computational power of constraint-based reasoning engines with the expressive power of first-order logic programming, thus expanding the range and size of planning problems that can efficiently be solved.

To wrap up — getting back from the technology level to the application level — there are dozens of features that are worth being integrated into planning, ranging from resource handling to configuration and design. But not only such "revolutionary" extensions should be considered. As planning will make its way into applications, many people will realize that every real-world problem is a bit different, and changes/additions to the engine will nearly always be necessary – promoting the use of planning approaches that are based on general solvers instead of highly specialized planning systems.

5.2 Extensions of Constraints Research

Motivated by needs in planning, constraints research will also be pushed to explore many new areas. Techniques like structural constraint satisfaction were already mentioned, but many other will get more attention for advanced planning features. The following list mentions a few:

- **Interactivity:**
  Planning customers may want to work interactively with their planning systems. Work on dynamic constraint satisfaction [Verfaillie94], which addresses changing problems, can support the corresponding evolving environment. Work on explanation of constraint reasoning [Sqalli96], which illuminates both success and failure, can support iterative evolution toward an acceptable alternative. Work on constraint solving advice can aid the user in responding to opportunities and difficulties.

- **Impreciseness:**
  Many planning problems involve unknown or imprecise information. Work on "partial", "soft" and "over-constrained" constraint satisfaction [Freuder92, Ruttkay94, Bistarelli99], which express preferences and uncertainty, can provide a richer language to express these planning needs. Stochastic constraint satisfaction [Walsh02], which enables to handle variables that follow some probability distributions, and approaches like structural constraint satisfaction [Nareyek01a] that enable to drop the the closed-world assumption, may provide important starting points here as well.

- **Distribution:**
  Planning customers may want to cooperate with other customers in joint planning operations. Planning can proceed in groups and through software agent representatives. The planning problems or the solution

process can be distributed. Increasingly customers will want to plan over the internet and through wireless communication. Work on distributed constraint satisfaction [Yokoo01] and on constraints and agents can support cooperative planning.

- **Ease of Use:**
  Planning customers will want software that can be used by non-specialists. Work on automating constraint modeling and synthesizing constraint solving will be useful here. Ease of use is desirable on a large scale, where the broader relevance of constraint satisfaction methods can assist in enterprise integration, from the design stage on through manufacturing, marketing, distribution, and maintenance. Ease of use is also desirable on a small scale, where personal planners can embody an understanding of their users in the form of constraints.

6 Conclusion

We hope that this article has provided a better understanding of the interplay of constraints and planning, and that it helped to increase awareness of constraint-based methods in the planning community as well as awareness of the action planning domain in the communities of constraint-based search. Bringing both together has great potential, scientifically as well as from an application point of view, and we would be happy if this article contributes to increased interaction of those communities.

Acknowledgments

We would like to thank the reviewers for many improvement suggestions, as well as Kate Larson and Vincent Conitzer for a final check.

The work described in this article is supported by a number of research sponsors and organizations. The authors and their sponsors are authorized to reproduce and distribute reprints for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained in this article should not be interpreted as views of the research sponsors or organizations involved.

Bibliography

[**Allen90**] Allen, J.; Hendler, J.; and Tate, A. (eds.) 1990. *Readings in Planning.* Morgan Kaufmann Publishers.

[**Beek99**] Van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), 585–590.

[**Bessière95**] Bessière, C.; Freuder, E. C.; and Régin, J.-C. 1995. Using Inference to Reduce Arc Consistency Computation. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), 592–598.

[**Biere99**] Symbolic Model Checking without BDDs. 1999. Biere, A.; Cimatti, A.; Clarke E. M.; and Zhu, Y. 1999. In Proceedings of the Fifth International Conference on Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), 193–207.

[**Bisschop82**] Bisschop, J. J. and Meeraus, A. 1982. On the Development of a General Algebraic Modeling System in a Strategic Planning Environment. Mathematical Programming Study 20: 1–29.

[**Bistarelli99**] Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; Verfaillie, G. 1999. Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. *Constraints* 4(3): 199–240.

[**Blum97**] Blum, A. L., and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. Artificial Intelligence 90: 281–300.

[**Boutilier00**] Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), 355–362

[**Currie91**] Currie, K. and Tate, A. 1991. O-Plan: The Open Planning Architecture. Artificial Intelligence 52(1): 49–86.

[**Davenport94**] Davenport, A.; Tsang, E.; Wang, C. W.; and Zhu, K. 1994. GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 325–330.

[**Davis60**] Davis, M., and Putnam, H. 1960. A Computation Procedure for Quantification Theory. *Journal of the ACM* 7(3): 201–215.

[**Dechter91**] Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint

Networks. *Artificial Intelligence* 49: 61–95.

[**Do00**] Do, B., and Kambhampati, S. 2000. Solving Planning Graph by Compiling it into a CSP. In Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000).

[**Erol94a**] Erol, K.; Hendler, J,; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 249–254.

[**Erol94b**] Erol, K.; Hendler, J,; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1123–1128.

[**Fikes71**] Fikes, R. E., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 5(2): 189–208.

[**Fourer83**] Fourer, R. 1983. Modeling Languages versus Matrix Generators for Linear Programming. ACM Transactions on Mathematical Software 9: 143–183.

[**Fourer03**] Fourer, R.; Gay, D. M.; and Kernighan, B. W. 2003. AMPL: A Modeling Language for Mathematical Programming, 2nd edition. Reading, Thomson Brooks/Cole, Pacific Grove, CA.

[**FrankTA**] Frank, J., and Jónsson, A. Constraint-based Attribute and Interval Planning. *Constraints*, Special Issue on Planning, to appear.

[**Freuder92**] Freuder, E. C., and Wallace, R. J. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58: 21–70.

[**Ginsberg96**] Ginsberg, M. L. 1996. Do Computers Need Common Sense? In Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96): 620–626.

[**Goldman00**] Goldman, R. P., Haigh, K. Z.; Musliner, D. J.; and Pelican, M. 2000. MACBeth: A Multi-Agent Constraint-Based Planner. In Papers from the AAAI-2000 Workshop on Constraints and AI Planning, Technical Report, WS-00-02, 11–17. AAAI Press, Menlo Park, California.

[**Gomes98**] Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting Combinatorial Search Through Randomization. In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), 431–437.

[**Han88**] Han, C., and Lee, C. 1988. Comments on Mohr and Henderson's

Path Consistency Algorithm. *Artificial Intelligence* 36, 125–130.

[**Hooker00**] Hooker, J.; Ottosson, G.; Thorsteinsson, E. S.; Kim, H.-J. 2000. A Scheme for Unifying Optimization and Constraint Satisfaction Methods. Knowledge Engineering Review 15(1): 11–30.

[**Jacopin00**] Jacopin, É, and Penon, J. 2000. On the Path from Classical Planning to Arithmetic Constraint Satisfaction. In Papers from the AAAI-2000 Workshop on Constraints and AI Planning, Technical Report, WS-00-02, 18–24. AAAI Press, Menlo Park, California.

[**Joslin96**] Joslin, D. 1996. Passive and Active Decision Postponement in Plan Generation. PhD thesis, University of Pittsburgh, Pittsburgh, PA.

[**Kautz92**] Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92), 359–363.

[**Kautz96**] Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 1194–1201.

[**Kautz98**] Kautz, H., and Selman, B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search, 58–60.

[**Kondrak97**] Kondrak, G., and van Beek, P. 1997. A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence* 89: 365–387.

[**Kuip93**] Kuip, C. A. C. 1993. Algebraic Languages for Mathematical Programming. European Journal of Operational Research 67(1): 25–51.

[**Laborie95**] Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), 1643–1649.

[**Laborie01**] Laborie, P. 2001. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. In Proceedings of the 6th European Conference on Planning (ECP-01).

[**Lever94**] Lever, J. M., Richards, B. 1994. parcPLAN: A Planning Architecture with Parallel Action, Resources and Constraints. In Proceedings of the Nineth International Symposium on Methodologies for Intelligent Systems (ISMIS 1994), 213–222.

[**Levesque97**] Levesque, H. J.; Reiter, R.; Lesprance, Y.; Lin, F.; and Scherl, R.B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. Journal of Logic Programming 31(1-3): 59–83.

[**Liatsos00**] Liatsos, V. 2000. Scaleability in Planning with Limited Resources. PhD Thesis, IC-Parc, Imperial College, submitted December 2000.

[**Milano00**] Milano, M.; Ottosson, G.; Refalo, P.; and Thorsteinsson, E. S. 2000. The Benefits of Global Constraints for the Integration of Constraint Programming and Integer Programming. In Working Notes of the AAAI-2000 Workshop on Integration of AI and OR Techniques for Combinatorial Optimization.

[**Minton92**] Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58: 161–205

[**Mittal90**] Mittal, S., and Falkenhainer, B. 1990. Dynamic Constraint Satisfaction Problems. In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), 25–32.

[**Muscettola94**] Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M., and Fox, M. S. (eds.), *Intelligent Scheduling*, Morgan Kaufmann, 169–212.

[**Moskewicz01**] Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In Proceedings of the 38th Design Automation Conference (DAC 2001), 667–672.

[**Nareyek01a**] Nareyek, A. 2001. Constraint-Based Agents – An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds. Reading, Springer LNAI 2062.

[**Nareyek01b**] Nareyek, A. 2001. Using Global Constraints for Local Search. In Freuder, E. C., and Wallace, R. J. (eds.), *Constraint Programming and Large Scale* Discrete Optimization, American Mathematical Society Publications, DIMACS Volume 57, 9–28.

[**Paolucci00**] Paolucci, M.; Shehory, O.; and Sycara, K. 2000. Interleaving Planning and Execution in a Multiagent Team Planning Environment. *Electronic Transactions on Artificial Intelligence, 4(2000), Section A, 23–43.* `http://www.ep.liu.se/ej/etai/2000/003/`*.*

[**Penberthy94**] *Penberthy, J. S., and Weld, D. S. 1994. Temporal Planning with Continuous Change. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1010–1015.*

[Refalo99] *Refalo, P. 1999. Tight Cooperation and Its Application in Piecewise Linear Optimization. In Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP99), 375–389.*

[Rintanen99] *Rintanen, J. 1999. Constructing Conditional Plans by a Theorem-Prover. Journal of Artificial Intelligence Research (10): 323–352.*

[Rintanen99] *Rintanen, J., and Jungholt, H. 1999. Numeric State Variables in Constraint-based Planning. In Proceedings of the Fifth European Conference on Planning (ECP-99).*

[Ruttkay94] *Ruttkay, Z. 1994. Fuzzy Constraint Satisfaction. In Proceedings of the Third IEEE International Conference on Fuzzy Systems, 1263–1268.*

[Sacerdoti75] *Sacerdoti, E. D. 1975. The Nonlinear Nature of Plans. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), 206–214.*

[Sadeh96] *Sadeh, N., and Fox, M. 1996. Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem. Artificial Intelligence 86, 1–41.*

[Selman92] *Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), 440–446.*

[Schuurmans00] *Schuurmans, D., and Southey, F. 2000. Local search characteristics of incomplete SAT procedures. In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), 297–302.*

[Sqalli96] *Sqalli, M. H., and Freuder, E. C. 1996. Inference-Based Constraint Satisfaction Supports Explanation. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 318–325.*

[Stefik81] *Stefik, M. J. 1981. Planning with Constraints (MOLGEN: Part 1). Artificial Intelligence 16(2): 111–140.*

[Tate77] *Tate, A. 1977. Generating Project Networks. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77), 888–893.*

[Tate94] *Tate, A.; Drabble, B.; and Kirby, R. 1994. O-Plan2: An Open Architecture for Command, Planning, and Control. In Zweben, M., and Fox, M. S. (eds.), Intelligent Scheduling, Morgan Kaufmann, 213–239.*

[**Tate00**] *Tate, A., Levine, J., Jarvis, P. and Dalton, J. 2000. Using AI Planning Technology for Army Small Unit Operations. Poster Paper in Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000), 379–386.*

[**Tsamardinos**] *Tsamardinos, I.; Vidal, T.; and Pollack, M. E. CTP: A New Constraint-Based Formalism for Conditional, Temporal Planning. Constraints, Special Issue on Planning, to appear in 2003.*

[**VanHentenryck99**] *Van Hentenryck, P. 1999. The OPL Optimization Programming Language. Reading, MIT Press.*

[**Verfaillie94**] *Verfaillie, G., and Schiex, T. 1994. Solution Reuse in Dynamic Constraint Satisfaction Problems. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 307–312.*

[**Vossen01**] *Vossen, T.; Ball, M.; Lotem A.; and Nau, D. 2001. Applying Integer Programming to AI Planning. Knowledge Engineering Review 16: 85–100.*

[**Yokoo01**] *Yokoo, M. 2001. Distributed Constraint Satisfaction - Foundations of Cooperation in Multi-agent Systems. Reading, Springer Series on Agent Technology.*

[**Walsh02**] *Walsh, T. 2002. Stochastic Constraint Programming. In Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI 2002).*

[**Wilkins88**] *Wilkins, D. 1988. Practical Planning: Extending the Classical AI Planning Paradigm. Reading, Morgan Kaufmann Publishers.*

[**Wolfman99**] *Wolfman, S. A., and Weld, D. S. 1999. The LPSAT Engine & its Application to Resource Planning. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), 310–316.*

[**Wolsey98**] *Wolsey, L. A. 1998. Integer Programming. Reading, Wiley.*