Constraints and AI Planning

Alexander Nareyek and Eugene C. Freuder, *University College Cork*

Robert Fourer, *Northwestern University*

Enrico Giunchiglia, *University of Genova*

Robert P. Goldman, *Smart Information Flow Technologies*

Henry Kautz, *University of Washington*

Jussi Rintanen, *Albert-Ludwigs-University Freiburg*

Austin Tate, *University of Edinburgh*

Tackling real-world planning problems often requires considering various types of constraints, which can range from simple numerical comparators to complex resources. This article provides an overview of techniques to deal with such constraints by expressing planning within general constraint-solving frameworks.

Our goal here is to explore the interplay of constraints and planning, highlighting the differences between propositional satisfiability (SAT), integer programming (IP), and constraint programming (CP), and to discuss their potential in expressing and solving AI planning problems.

Interest in constraint techniques for AI planning problems has grown considerably in recent years. 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 of constraint-based frameworks are *constraints* and *variables*; a constraint is an entity that restricts the values of variables. Such frameworks can easily express not only simple orderings in partial-order planning systems but also complex problem features, such as scheduling and reasoning about numerical resources. The exclusion relations of Graphplan-based planners can also be interpreted in this context, and recently researchers have even developed planning systems that can fully cast planning problems as a constraint satisfaction problem.

However, to remain efficient, most constraint-based search frameworks use only a restricted scenario. SAT, for example, restricts constraints to propositional formulas and constrains variables to a Boolean domain. Similarly, integer linear programming applies linear inequalities to restrict numerical variables. CP, on the other hand, is more general; it doesn't restrict the types of constraints (although usually only variables with finite domains are considered).

The basic planning problem

A basic planning problem usually comprises an initial world description, a partial description of the goal world, and a set of actions (sometimes also called *operators*) that map a partial world description to another. The problem can be enriched by including additional aspects, such as temporal or uncertainty issues, or by requiring the optimization of certain properties. A solution is a sequence of actions leading from the initial world description to the goal world description, referred to as a *plan*.

We assume readers are familiar with basic planning techniques and terminology (for more information, see *Readings in Planning*[1]). (Also, note that we don't cover specialized reasoning schemes, such as the management of temporal constraint networks.[2])

Planning as propositional satisfiability

SAT determines whether a propositional formula is satisfiable. Such a formula consists of two-valued variables (true and false), related through the operators ¬ ("not"), ∨ ("or" or disjunction), and ∧ ("and" or conjunction). Most solvers require that we state the problem as a conjunctive normal form (a conjunction of disjunctions), and SAT-solving techniques include refinement approaches such as Satz-Rand[3] and Chaff[4] (which are based mostly on the Davis-Putnam procedure[5]) and local-search methods such as GSAT[6] and SDF.[7]

Planning as satisfiability is a paradigm that Henry Kautz and Bart Selman originally proposed.[8] The idea is simple. Suppose we have a finite description of the planning problem; more specifically, assume a STRIPS description D[9] with finitely many objects and time steps. Because the domain is finite, we can write a propositional formula TR_i whose models are one-to-one with the possible transitions in D. There are several ways to write such a TR_i.[10] For our goals, it suffices to say that TR_i has a propositional variable A_i for each ground action A and two propositional variables F_i and F_{i+1} for each ground fluent F in D (*fluent* is equivalent to a state variable). Intuitively, F_i represents the value of F at time i (and similarly for A_i and F_{i+1}).

If D is deterministic and we're given a single initial state represented by a formula I, we can find a valid plan that reaches a goal state represented by a formula G in n steps if we can satisfy

$$I_0 \wedge \left(\bigwedge_{0 \le i < n} TR_i \right) \wedge G_n \qquad (1)$$

where 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 idea has led to impressive results and has thus greatly influenced the planning community.[10]

For very large values of n (say, exponential in the size of D), Equation 1's size becomes problematic, and planning as satisfiability might not be feasible. However, if we look for short plans, it can be a winning approach. Researchers have obtained similar considerations and results for formal verification, where TR_i encodes the transition relation of the system under analysis, $\neg G$ is the property needing verification, and I represents the set of possible initial states. It's possible to violate the property in the first n ticks if Equation 1 is satisfiable, and systems based on this idea are far more effective than the others (if n is small).[11]

Given this, the planning-as-satisfiability approach clearly isn't restricted to work with STRIPS domain descriptions. We can start with a domain description written in any language for specifying domains. Then we can determine the plans of length n by satisfying Equation 1 as long as the domain is deterministic, a single initial state exists, and propositional formula TR_i encodes all the possible transitions.

Using constraints and related techniques as an underlying framework for problem-solving tasks—such as planning—has proven successful for many domains.

Operations research approaches to AI planning

The concept of an *operations research* approach has long been influential in AI planning. In fact, OR methods inspired much of the early work on heuristic search. For example, the Nonlin hierarchical partial-order planner[12] was developed within a project entitled "Planning: A Joint AI/OR Approach," and it incorporated OR methods for temporal-constraint satisfaction. The ZENO temporal planner adopted OR data structures and algorithms for handling linear-inequality constraints.[13] If we broadly interpret "operations research" as the study of operations, then it can encompass almost any AI planning method.

Nevertheless, for highly combinatorial problems such as AI planning, a distinctive IP approach exists that's 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. We can then find a solution using a general-purpose branch-and-bound procedure, based on the idea of a search tree.[14] The branching of the tree corresponds to dividing the original problem into progressively more constrained subproblems, with leaf nodes representing individual solutions. A full tree search would thus involve a complete enumeration. However, 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 subproblems, in conjunction with a variety of ingenious heuristic procedures.

Classical AI planning problems don't require minimization or maximization, and we can instead use the branch-and-bound procedure to seek a feasible solution to the constraints. Still, the search for a solution might be guided by constructing an objective such as minimization of the number of actions taken.

An example of an IP approach for planning

Thomas Vossen and his colleagues report on AI planning experiments that are representative of using the IP approach for combinatorial optimization.[15] They consider first a straightforward formulation, based on the SAT representation that Blackbox[16] employs, in terms of integer variables that take the value of 1 or 0, depending on whether a certain condition holds:

- Do[a,t] = 1 if and only if action a is taken in time step t
- True[f,t] = 1 if and only if fluent (assertion) f is true in time step t

Linear constraints on these variables ensure that state transitions are consistent with actions. Because the variables take 0 or 1 values rather than SAT's true or false values, the IP's operators are arithmetic rather than logical. However, the assertions that the IP's constraints express correspond directly to the logical conditions in the SAT representation.

As a concrete example, consider a case of a blocks world having an arm that can 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 time step 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 time step 3 unless it has performed one of the three actions compatible with holding it at time step 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 time step, 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 the final state's requirements. We can mechanically derive all the necessary constraints of these types from the contents of the problem's STRIPS representation.

We need to enforce only the Do variables' integrality. This is because the True variables might simply be constrained to lie in the interval [0, 1], in which case the constraints will force them to take only the values of 0 or 1 at an optimal solution. Even so, a highly respected branch-and-bound implementation failed to prune the search tree sufficiently to avoid prohibitive increases in execution time and memory requirements for over half of the test problems considered.[15]

Using a tighter representation

As is often the case in IP, we can produce much better results using a "tighter" representation that's less intuitive but yields better bounds for pruning the search tree. One such representation replaced the True variables with five collections of variables similarly indexed over fluents and time steps:[15]

- Do[a,t] = 1 if and only if action a is taken in time step t
- Keep[f,t] = 1 if and only if f is kept true in time step t
- Add[f,t] = 1 if and only if f isn't a precondition but is added in time step t
- PreAdd[f,t] = 1 if and only if f is a precondition but isn't deleted in time step t
- PreDel[f,t] = 1 if and only if f is a precondition and is deleted in time step t

For each fluent and time step, three constraints define the relations between the new variables. So, 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 time step 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, Keep, PreAdd, or PreDel apply only if one of Keep, PreAdd, or Add applied at the previous time step:

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

By providing high-level, symbolic expressions for indexed collections of constraints, modeling languages and their supporting environments encourage experimentation with a variety of formulations.

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

```
Add['hold_B',3] ≥ Do['pick_up_B',3]
Add['hold_B',3] ≥ Do['unstack_B_A',3]
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 the problems considered (though with computation times over 1,000 seconds and search trees of over 80,000 nodes for the two hardest cases). Even so, Blackbox, which combines more specialized processing of AI planning problems with SAT solvers, solved conventional test problems even more efficiently.

General-purpose IP software's effectiveness has improved considerably in the few years since the tests just described were run. IP might also be more suitable for temporal or resource-oriented planning problems, because these involve larger numerical variable domains that are hard to encode using SAT approaches. Another alternative is to combine solvers of both types for those tasks. Steven Wolfman and Daniel Weld developed a system that uses a SAT solver to solve the core planning problem and uses IP to address possible numerical relations.[17]

Modeling languages

IP's flexibility has been further strengthened by the development of several modeling languages[18–20] that let us write general formulations in a way that's concise and natural for modelers yet can be efficiently processed by computer. For example, we can write the collection of all the constraints in the previous subsection, enforcing the relationship between the Add and the Do variables for all combinations of fluents and time steps, in AMPL[21] 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. For example, the sum operator denotes the mathematical Σ, character pairs >= and <= represent the inequality relations ≥ and ≤, 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 encourage experimentation with a variety of formulations.

Applying constraint programming to planning

The CP approach is characterized by more natural (or convenient) formulations than SAT or IP. Problems are described as *constraint satisfaction problems* (CSPs), which consist of a set of variables $x = \{x_1, \ldots, x_n\}$, where each variable is associated with a domain $d_1, \ldots, d_n$ and a set of constraints $c = \{c_1, \ldots, c_m\}$ over these variables.

In a CSP, the domains can be symbols as

Constraint Programming Search Techniques

Once we model a problem, we can employ different search techniques to extract a solution. The most common is a tree-based refinement search, which, unlike branch-and-bound (see the main article), doesn't apply continuous relaxations. Instead, it applies at all decision nodes a variety of branch selection, domain tightening, and simplification procedures to reduce the extent of the tree that must be searched.

The first central component of refinement search is variable/value selection techniques, which select a variable to which a specific value is assigned and then determine the value itself. This assignment process is called labeling and iterates until it has assigned values to all variables. Researchers have proposed numerous variable and value-ordering heuristics, such as smallest-domain-first or smallest-value-first.[1] Inconsistencies (that is, when a constraint is violated) trigger backtracking.[2]

The second central component is propagation or consistency techniques. After each labeling step, reductions of the other variables' domains can be deduced, which might 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[3] and PC-4.[4]

A special type of constraint is the *global constraint*. These higher-level constraints usually provide functionality to perform highly specialized and very efficient propagations.

A different paradigm to search for solutions is that of local search, which doesn't stepwise reduce the variables' domains but changes concrete assignments for the variables iteratively. However, for constraint programming, search techniques based on local search aren't popular so far. This might be due to the traditional logic-programming framework for constraint programming. Nevertheless, researchers have proposed many methods, including the min-conflicts heuristic,[5] GENET,[6] and an approach based on global constraints.[7]

References

1. N. Sadeh and M. Fox, "Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem," *Artificial Intelligence*, vol. 86, no.1, 1996, pp. 1–41.
2. G. Kondrak and P. van Beek, "A Theoretical Evaluation of Selected Backtracking Algorithms," *Artificial Intelligence*, vol. 89, nos. 1–2, 1997, pp. 365–387.
3. C. Bessière, E.C. Freuder, and J.-C. Régin, "Using Inference to Reduce Arc Consistency Computation," *Proc. 14th Int'l Joint Conf. Artificial Intelligence* (IJCAI 95), 1995, pp. 592–598.
4. C. Han and C. Lee, "Comments on Mohr and Henderson's Path Consistency Algorithm," *Artificial Intelligence*, vol. 36, no. 1, 1988, pp. 125–130.
5. S. Minton et al., "Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems," *Artificial Intelligence*, vol. 58, no. 1–3, 1992, pp. 161–205.
6. A. Davenport et al., "GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement," *Proc. 12th Nat'l Conf. Artificial Intelligence* (AAAI 94), AAAI Press, 1994.
7. A. Nareyek, "Using Global Constraints for Local Search," *Constraint Programming and Large Scale Discrete Optimization*, E.C. Freuder and R.J. Wallace, eds., Amer. Mathematical Soc. Publications, DIMACS, vol. 57, 2001, pp. 9–28.

well as numbers and can be continuous or discrete (such as "block A," "13," or "6.5"). Constraints are relations between variables (for example, "x_a is on top of x_b" or "$x_a < x_b \times x_c$") 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 (see the "Constraint Programming Search Techniques" sidebar).

We can represent the CSP structure of variables and constraints 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 as a *constraint graph* instead of a CSP in the following.)

A comparison with SAT and IP

The CP framework's advantages become apparent in slightly more complex settings, such as when incorporating resources and time. Consider, for example, a scenario in which some tasks with specific durations share a resource, and the goal is to prevent overlap. IP's mathematical framework and SAT's propositional formulas are highly restricted, so a problem must be translated into a very specific form, which often requires an expert.

For the SAT approach, representing the numerical aspects of the task durations of the nonoverlap example is hardly possible because the propositional SAT variables can represent only qualitative differences. This differs from CP and IP, which can also represent quantitative information. Of course, we can also model discrete numbers in SAT 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 it destroys the information about the values' ordering relation, which should be exploited during search. For an IP approach, the inherent disjunction in the nonoverlap problem (task A may precede task B, or task B may precede task A) blows up the size of a "representation by inequalities" enormously.

CP, on the other hand, lets us use higher-level (global) constraints. A formulation could simply look like **nonoverlap(taskA_begin, taskA_duration, taskB_begin, taskB_duration)**. Add-on languages such as AMPL for IP relax this statement only slightly because many problem aspects can't be adequately translated to IP's linear inequalities. Furthermore, global constraints don't just simplify modeling—they can also capture domain-specific dependencies. The IP and SAT approaches can break down problems into their specific frameworks and then scan the resulting specifications for structures on which to apply specific solution strategies. However, even though researchers have developed many efficient methods—SAT's propositional clauses and IP's linear inequalities can scarcely exploit the higher-level domain knowledge to support search.[22] CP's global constraints and corresponding solution techniques have been an important factor in its success.

Another advantage of using CP for planning is that numerous planning decisions with discrete alternatives exist, and OR methods' performance declines sharply as the number of integer variables in the problem increases. However, efforts to combine CP with IP are ongoing.[22–24]

Constraint-based planners

We adopt a model-based viewpoint here, defining constraint-based planners by the way the planning problem is cast—using explicit constraints. These constraint types aren't bound to propositional clauses or linear inequalities.

By contrast, a widely used solution method for CSPs—propagation techniques—are sometimes seen as CP's essential ingredient, and planners applying this technique in one form or another are called constraint-based. From this viewpoint, 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 first appearance of a system involving constraints was the MOLGEN planner.[25] More recent planners based on this idea include Descartes,[26] parcPLAN,[27,28] CPlan,[29] the approach of Jussi Rintanen and Hartmut Jungholt,[30] GP-CSP,[31] the approach of Eric Jacopin and Jacques Penon,[32] MACBeth,[33] the EXCALIBUR agent's planning system,[34] EUROPA,[35] CPplan,[36] and CSP-PLAN.[37]

We group constraint-based planners into three categories. The first one, *planning with constraint posting*,

- uses CP for subproblems of planning and
- exhibits limited interaction between CSP solving and the actual planning process.

The second, *planning with maximal graphs*,

- constructs a large CSP, involving all possible planning options up to a specific plan size,
- exhibits full interaction between value- and structure-based problem aspects, and
- doesn't scale well.

The last category, *completely capturing planning within CP*,

- expresses the complete planning problem with the CP framework,
- requires an extended CP framework to cover different possible graph structures, and
- exhibits full interaction between value- and structure-based problem aspects.

Planning with constraint posting. This approach uses CP for planning by posting or adding (or retracting, in the case of backtracking) constraints during a conventional planning process. It uses constraint satisfaction as an add-on for planning to check the satisfaction of restrictions such as numerical relations. For example, MOLGEN, MACBeth, and Eric Jacopin and Jacques Penon's planning procedure apply this scheme. A more integrated approach is implemented in the Descartes and *parc*PLAN systems, which use constraint postings not only for numerical values but also for postponing some action-choice decisions. In a wider context, systems such as I-Plan/O-Plan,[38–40] IxTeT,[41] and HSTS[42] also fall into this category.

Hierarchical task networks provide a natural way to stipulate global constraints on plans, meshing well with the needs of systems that combine planning and constraint satisfaction.

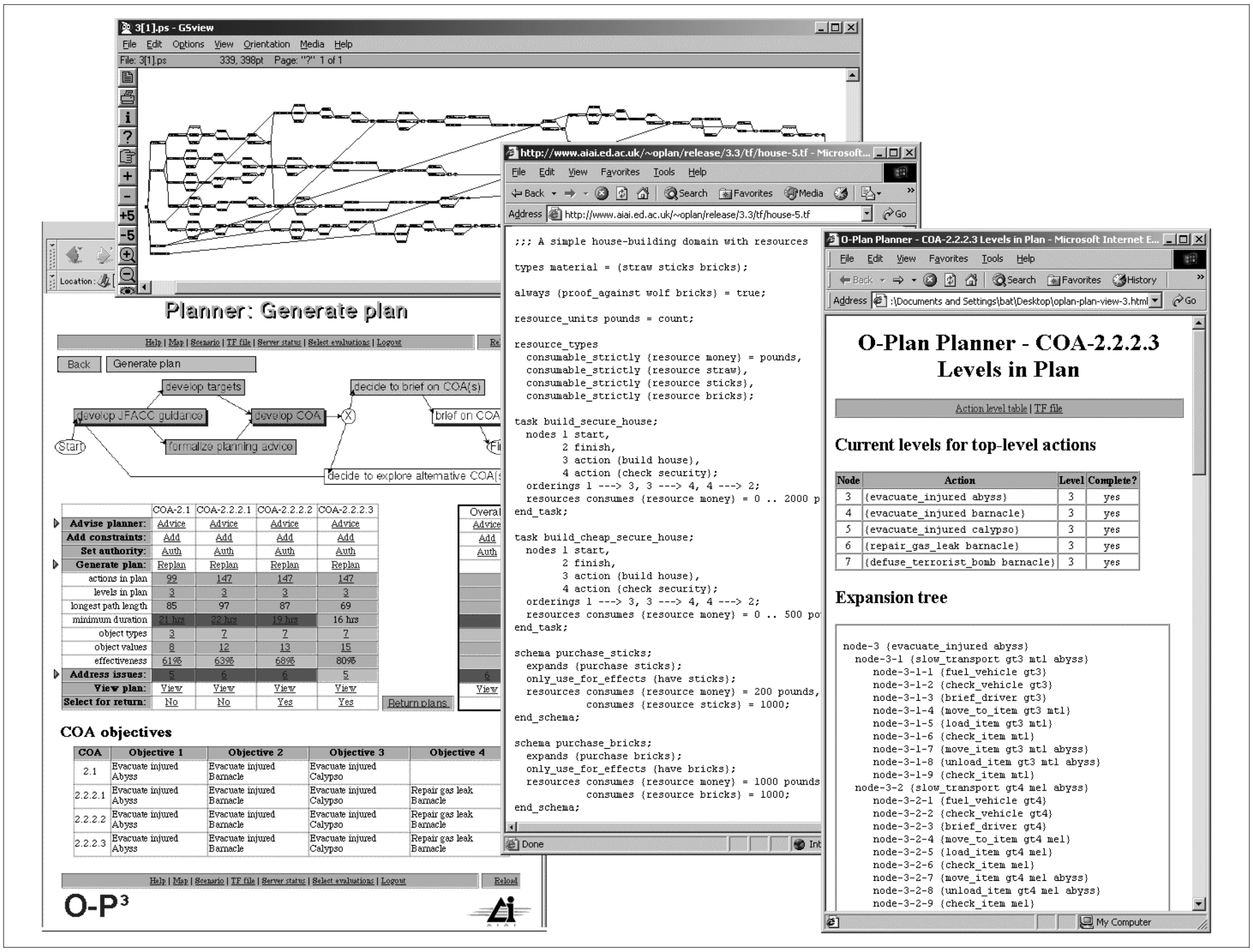
Integrating constraint posting into *hierarchical task network* planning[43] (sometimes called *decomposition* planning) often provides an alternative to the more conventional means-and-ends or "first principles" planning. In particular, HTN planning is useful when planning problems apply standard operating procedures (or other well-established process fragments) rather than the "puzzle mode" planning that characterizes domains such as the blocks world. Besides plan generation, HTN systems also address the problem of modeling domains in which planning occurs. They make it easy to understand, control, and communicate tasks, plans, intentions, and effects between agents.[44] They also let users and computer systems cooperate using a mixed-initiative style.[33,40] 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.[45]

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, and although they apply previously known procedures, they're not 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 might be available only through ancillary black-box problem solvers.

Researchers have used an HTN least-commitment planning approach for many years in practical planning systems such as NOAH (Nets of Action Hierarchies),[46] Nonlin,[12] and SIPE.[47] Two existing planners that combine HTN planning and constraint satisfaction techniques are the Artificial Intelligence Applications Institute'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 for employing powerful problem solvers based on search and constraint-reasoning methods while still retaining human intelligibility of the overall planning process and products.

A tactical planner's main job is to select and map appropriate procedures or processes onto different situations. In the process, the planner must explore and compare multiple options and let users do the same (see Figure 1). It must also 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 planning, both to provide heuristic guidance ("I suggest first trying to fly in formation during the mission's ingress phase") and constraints ("You must fly over this point").

HTN planners differ from conventional first-principles planners in how they generate subgoals. First-principles planners generate subgoals to satisfy the preconditions of the operators they want to execute. For example, in the blocks world, the desire to stack block A on block B might cause a first-principles planner to generate a subgoal to clear block B. Subgoaling continues until all the subgoals have been satisfied by adding operators or by the initial conditions (in our example, if block B is already clear in the starting state). On the other hand, an HTN planner decomposes complex goals into simpler goals, building a tree (or graph, if we can use operators for more than one purpose) from the initial complex goals down to leaves that are all executable primitives. At each level, we can post or add new constraints to refine the feasibility testing.


3[1].ps - GSview
File Edit Options View Orientation Media Help
File: 3[1].ps 339, 398pt Page: "?" 1 of 1
Location:
Planner: Generate plan
Help | Map | Scenario | TF file | Server status | Select evaluations | Logout
Back
Generate plan
develop targets
decide to brief on COA(s)
develop JFACC guidance
develop COA
brief on COA
Start
formalize planning advice
decide to explore alternative COA(s
Return plans
Help | Map | Scenario | TF file | Server status | Select evaluations | Logout
Reload
O-P³
http://www.aiai.ed.ac.uk/~oplan/release/3.3/tf/house-5.tf - Microsoft...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.aiai.ed.ac.uk/~oplan/release/3.3/tf/house-5.tf
Done
O-Plan Planner - COA-2.2.2.3 Levels in Plan - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Search Favorites History
Address :\Documents and Settings\bat\Desktop\oplan-plan-view-3.html
Go
Action level table | TF file
My Computer

	COA-2.1	COA-2.2.2.1	COA-2.2.2.2	COA-2.2.2.3
Advise planner:	Advice	Advice	Advice	Advice
Add constraints:	Add	Add	Add	Add
Set authority:	Auth	Auth	Auth	Auth
Generate plan:	Replan	Replan	Replan	Replan
actions in plan	99	147	147	147
levels in plan	3	3	3	3
longest path length	85	97	87	69
minimum duration	21 hrs	22 hrs	19 hrs	16 hrs
object types	3	7	7	7
object values	8	12	13	15
effectiveness	61%	63%	68%	80%
Address issues:	5	6	6	5
View plan:	View	View	View	View
Select for return:	No	No	Yes	Yes

Overal
Advice
Add
Auth
6
View

COA objectives

COA	Objective 1	Objective 2	Objective 3	Objective 4
2.1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	
2.2.2.1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle
2.2.2.2	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle
2.2.2.3	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle

```
;;; A simple house-building domain with resources

types material = (straw sticks bricks);

always {proof_against wolf bricks} = true;

resource_units pounds = count;

resource_types
  consumable_strictly {resource money} = pounds,
  consumable_strictly {resource straw},
  consumable_strictly {resource sticks},
  consumable_strictly {resource bricks};

task build_secure_house;
  nodes 1 start,
        2 finish,
        3 action {build house},
        4 action {check security};
  orderings 1 ---> 3, 3 ---> 4, 4 ---> 2;
  resources consumes {resource money} = 0 .. 2000 p
end_task;

task build_cheap_secure_house;
  nodes 1 start,
        2 finish,
        3 action {build house},
        4 action {check security};
  orderings 1 ---> 3, 3 ---> 4, 4 ---> 2;
  resources consumes {resource money} = 0 .. 500 po
end_task;

schema purchase_sticks;
  expands {purchase sticks};
  only_use_for_effects {have sticks};
  resources consumes {resource money} = 200 pounds,
            consumes {resource sticks} = 1000;
end_schema;

schema purchase_bricks;
  expands {purchase bricks};
  only_use_for_effects {have bricks};
  resources consumes {resource money} = 1000 pounds
            consumes {resource bricks} = 1000;
end_schema;
```

O-Plan Planner - COA-2.2.2.3 Levels in Plan

Current levels for top-level actions

Node	Action	Level	Complete?
3	{evacuate_injured abyss}	3	yes
4	{evacuate_injured barnacle}	3	yes
5	{evacuate_injured calypso}	3	yes
6	{repair_gas_leak barnacle}	3	yes
7	{defuse_terrorist_bomb barnacle}	3	yes

Expansion tree

```
node-3 {evacuate_injured abyss}
  node-3-1 {slow_transport gt3 mtl abyss}
     node-3-1-1 {fuel_vehicle gt3}
     node-3-1-2 {check_vehicle gt3}
     node-3-1-3 {brief_driver gt3}
     node-3-1-4 {move_to_item gt3 mtl}
     node-3-1-5 {load_item gt3 mtl}
     node-3-1-6 {check_item mtl}
     node-3-1-7 {move_item gt3 mtl abyss}
     node-3-1-8 {unload_item gt3 mtl abyss}
     node-3-1-9 {check_item mtl}
  node-3-2 {slow_transport gt4 mel abyss}
     node-3-2-1 {fuel_vehicle gt4}
     node-3-2-2 {check_vehicle gt4}
     node-3-2-3 {brief_driver gt4}
     node-3-2-4 {move_to_item gt4 mel}
     node-3-2-5 {load_item gt4 mel}
     node-3-2-6 {check_item mel}
     node-3-2-7 {move_item gt4 mel abyss}
     node-3-2-8 {unload_item gt4 mel abyss}
     node-3-2-9 {check_item mel}
```

Figure 1. Interaction with the O-Plan system. A Web interface allows for the generation of multiple options for a plan. Other screens show the domain model in use and graphical and text-based tree views of a plan.

For example, an HTN planner for military air missions might start with a "perform reconnaissance" top-level goal, whose parameters indicate which location to survey. Matching this goal would be a task network (or method) that would decompose the top-level goal into three subgoals: "perform ingress," "overfly objective," and "perform egress." Additional constraints would indicate that we need to achieve the three subgoals in the order listed and that

- the ingress phase should bring the aircraft to an appropriate staging location,
- the overflying phase should carry the aircraft from the staging location to over the target, and
- the egress phase should carry the aircraft back to the base from the target area.

We could also add constraints for resource use, perhaps to ensure that the aircraft doesn't exceed its fuel allowance or fail to complete its mission by a given deadline.

HTN planning is particularly suited to planning with constraints because it makes it convenient to specify constraints that cover a plan's subspaces. For example, it's trivial to create a plan for air travel that specifies that the same airport must be used for departure and return. This is nontrivial for a conventional STRIPS-style planner, particularly if the agent's initial position and desired ending points aren't at the airport. Indeed, it's 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. 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 you want to take two flights, or four, or eight, possibly with multiple agents whose plans are interleaved.

It's also easy for an HTN planner to represent resource use over a plan's sections. 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 flight's three phases would provide further fuel use equations. This would make for efficient constraint propagation even when planning occurs at arbitrary points in the chronological sequence. For example, if a reconnaissance aircraft had 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 that constraint's effects over the entire plan. In contrast, conventional STRIPS-style planners have difficulty reasoning about such resources except from plan prefixes or suffixes. The

HTN planner's 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 the application of constraint-posting techniques because its expressive power makes it easy to specify global constraints and make them available to constraint solvers.

Planning with maximal graphs. In contrast to the approach discussed in the previous section—in which constraints were stepwise posted or added depending on the current plan refinement state—the techniques described here require posting all the domain's constraints at once. Similar to SAT and IP approaches, a restricted planning problem is encoded as conditional CSP (formerly called "dynamic" CSP,[48] but since what many people understand as a dynamic CSP is different, we use the term "conditional" here). The CSP is constructed such that it includes all possible plans up to a certain number of actions. The solving process can activate or deactivate the CSP's actions and related subparts, which can be interpreted as SAT's true and false values for the ground actions or fluents (0 or 1 values for IP). More compact representations are also possible. We say such approaches are based on *maximal graphs and structures*, because they require constructing a representation that encompasses all possible options.

Planning systems such as Jussi Rintanen and Hartmut Jungholt's system, CPlan, GP-CSP, CPplan, and CSP-PLAN follow this line. The advantage of constructing a structure with all possibilities is that we can easily propagate decisions throughout the whole CSP and can eliminate future decision options that become infeasible. For planning problems, we don't know the optimal size (for example, the number of actions) in advance, so the solving process must perform a stepwise extension of the CSP structures if we can't find a solution. Similar to SAT and IP formulations, which we can also classify as approaches based on maximal graphs, maximal structures can soon result in an unmanageable CSP size if the planning problem is slightly larger than simple toy problems.

Following the approach of maximal graphs, a CP variation on the formulations discussed earlier (in the IP section) could define fewer Do variables but with larger domains, taking members of the set of actions (rather than 0 or 1) as their values:

Do[t] = the action performed at time step t

Also, a CP model can directly express logical conditions, such as set membership and equivalence, that would have to be translated to numerical inequalities for IP. For example, suppose we write ADD[f] and PRE[f] for the subsets of actions that add fluent f and that have fluent f as a prerequisite, respectively. Then, the three IP constraints that relate the Add and Do variables for holding block B at time step 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']
```

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

Many modeling languages for CP, such as OPL (Optimization Programming Language),[49] can conveniently express these constraints. Consider again the relationship between the Add and Do variables. We can write the entire collection of constraints that enforce this relationship, for all combinations of fluents and time steps, 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];
```

For the best results when actually solving this CSP, the search should focus on the Do variables, because they effectively define all the others. However, someone familiar with the model might have to explicitly convey this information to the solver.

Completely capturing planning within constraint programming. Approaches that use maximal graphs might be suitable if the problem is small and only requires a finite number of variables (having NP complexity). As the complexity of planning problems increases, representation becomes difficult or impossible (this is also true for SAT and IP approaches). Such computational limitations arise, for example, in the 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 appears to be no natural way of restricting these problems to force them to NP. For example, probabilistic planning with the same plan size restriction is complete for NP^{PP}. This means we can't generate CSP representations for these problems in polynomial time.

For many simple cases, if the maximal-graph approach applies, the CSP graph size, even when polynomial, might not be practical. This is because even a linear increase in the number of variables in the representation might translate to an exponential increase in the time needed to solve the problem. Even more 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. This doesn't fit with the general idea of CP, which is that we shouldn't mix a problem specification with search control (which is one reason why CP approaches are so easily adapted and widely applicable).

The main problem for approaches that try to completely capture planning within CP is that the CP paradigm must somehow be extended such that the CSP graph can change dynamically instead of being built a priori. Earlier, we presented an approach that built the CSP graph stepwise through external planning. However, only a subproblem of planning is covered within CP, separating value- and structure-based problem aspects. In another approach,[34] an extended CP framework is used to seamlessly integrate the satisfaction and optimization of both aspects.

This approach doesn't create the CSP structure in advance to cover all possible plans. *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 optimization. The search can seamlessly interleave the satisfaction or optimization of variables' values and graph structure, and search can freely move around in the full planning problem's search space. In addition, the approach lets us drop the closed-world assumption (that is, we're no longer restricted to handle only a predefined number of objects in the world). The approach was developed for the EXCALIBUR agent's planning system, which uses a local-search approach to solve such problems.

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 a specific property's state, is always connected to exactly one action constraint. An action constraint ensures that connected precondition, operation, and state-change tasks form a valid action. If this structural constraint holds for all state-change tasks in a constraint graph (and potential other structural constraints hold as well), the graph is *structurally consistent* (see Figure 2a). Figure 2b shows a part of a constraint graph that's structurally inconsistent because the state-change task of adding 100 units of money at time 692 isn't part of an action—that is, it isn't connected to an action constraint as required by the structural constraint.

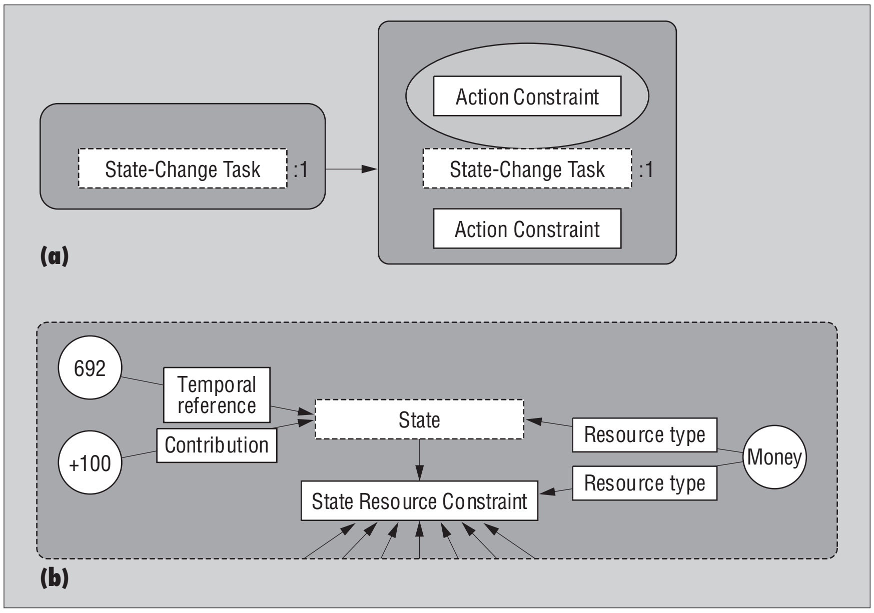


Figure 2. Extending the constraint-programming framework by structural constraints: (a) A structural constraint. The constraint applies at any part of the graph where the left side matches and is fulfilled if the right side matches then as well (the right side's dark area describes a "negative" match—that is, exactly one action constraint is required to be connected to the state-change task). (b) A partial view of a structurally inconsistent constraint graph. The structural constraint in (a) matches the state-change task but doesn't find a connected action constraint.

The inconsistency resolution (and optimization) of the EXCALIBUR planning system's local-search-based approach changes not only variable values but also the constraint graph. For example, a state resource constraint, which projects a specific property's state over time and checks if related precondition tasks are fulfilled, might add a new state-change task to the constraint graph to resolve an unsatisfied precondition. The previously described structural constraint will consequently be unsatisfied because the new state-change task isn't connected to any action constraint (as in Figure 2b). The search might then add a new action constraint to regain structural consistency.

Jeremy Frank and Ari Jónsson have presented a similar approach in the direction of completely capturing planning within CP.[35] Their approach is less general and focuses more on specific interval elements than general structural constraints. They define *compatibilities*, which represent implicative relations among interval elements and make it possible to require further intervals and constraints in specific situations.

Future directions

Constraint techniques have great potential for AI planning. One of the reasons is that using more general modeling techniques for planning makes integrating additional problem aspects easier.

Feature integration

Constraint-based methods' principal advantage lies in their generality—but this is also their principal disadvantage. We can introduce many kinds of additional requirements into the AI planning problem, such as in the case of IP, by simply adding linear inequalities to an integer program and reapplying the general branch-and-bound solver. Comparable additions to specialized planning systems tend to necessitate more substantial changes.

An example of incorporating additional problem aspects is the area of integrating planning with resource and scheduling features. Complex numerical and logical relations make adding such features in conventional planning systems difficult. In 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 solvers' open nature enables easy integration. An overview of related solving techniques appears elsewhere.[50]

Another interesting direction is the synthesis of constraint-based and first-order planning technology. Researchers have developed a large body of research based on turning first-order logic into a practical programming tool. GOLOG and its successors pushed the expressive power of logic-based planning and control languages.[51] 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, researchers have added the ability to handle quantitative probabilistic information and decision-theoretic control.[52]

GOLOG's greater expressive power comes at a computational cost: although it includes nondeterministic choice operators, it must use those operators with great care to avoid a combinatorial explosion. GOLOG's inference engine is simple Prolog-style chronological backtracking, but in practice the GOLOG user must write a nearly deterministic logic program. In short, GOLOG excels at representing open-ended 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 might build upon the insight that at a high-enough level of abstraction, many planning domains are essentially propositional. We could use algorithms based on Graphplan[53] and satisfiability testing to search candidate abstract plans' large state spaces. We could then use first-order-logic programming and decision-theoretic techniques to expand the abstract plans to executable specifications.

A complementary approach to synthesizing the two would be to build on Matthew Ginsberg's suggestion that the role of "commonsense knowledge" is to transform a real-world problem instance into a small propositional core that a brute-force search could solve.[54] 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 we can efficiently solve.

Moving back to the application level, dozens of features exist that are worth integrating into planning, ranging from resource handling to configuration and design. Using general constraint technology makes such integration easy—and not just for major extensions such as resource reasoning. As planning moves into applications, many people will realize that every real-world problem is slightly different and will almost always require changing the engine and making additions. This in turn will promote the use of planning approaches that are based on general solvers rather than highly specialized planning systems.

Extending constraints research

Motivated by planning needs, constraints research will explore many new areas. One new technique for advanced planning features is *interactivity*—planning customers might want to work interactively with their planning systems. Work on dynamic constraint satisfaction,[55] which addresses changing problems, can support the corresponding evolving environment. Work on generating explanations of constraint reasoning,[56] illuminating its success and failure, can support iterative evolution toward an acceptable alternative. Work on constraint-solving advice can help the user respond to opportunities and difficulties.

Another area is *impreciseness*. Many planning problems involve unknown or imprecise information. Work on partial, soft, and overconstrained constraint satisfaction,[57–59] which expresses preferences and uncertainty, can provide a richer language to express these planning needs. Important starting points might include stochastic constraint satisfaction,[60] which lets us handle variables that follow some probability distributions, and approaches such as structural constraint satisfaction[34] that let us drop the closed-world assumption.

Distribution is also important. Planning customers might 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[61] and on constraints and agents can support cooperative planning.

As planning moves into applications, many people will realize that every real-world problem is slightly different and will almost always require changing the engine and making additions.

Finally, *ease of use* will be important because planning customers will want software that nonspecialists can use. 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 design 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.

We hope we were able to shed some light on the interplay of constraints and planning. We also hope we've increased awareness of constraint-based methods in the planning community and of the action-planning domain in the communities of constraint-based search. Bringing both together has great potential, scientifically as well as from an applications viewpoint, and we encourage increased interaction between the two communities.

Acknowledgments

We thank the reviewers, editors, and Kate Larson and Vincent Conitzer for their many suggestions.

Many research sponsors and organizations support the work described in this article. 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.

References

1. J. Allen, J. Hendler, and A. Tate, eds., *Readings in Planning*, Morgan Kaufmann, 1990.
2. R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Networks," *Artificial Intelligence*, vol. 49, nos. 1–3, 1991, pp. 61–95.
3. C. Gomes, B. Selman, and H. Kautz, "Boosting Combinatorial Search through Randomization," *Proc. 15th Nat'l Conf. Artificial Intelligence* (AAAI 98), AAAI Press, 1998, pp. 431–437.
4. M. Moskewicz et al., "Chaff: Engineering an Efficient SAT Solver," *Proc. 38th Design Automation Conf.* (DAC 01), IEEE CS Press, 2001, pp. 667–672.
5. M. Davis and H. Putnam, "A Computation Procedure for Quantification Theory," *J. ACM*, vol. 7, no. 3, 1960, pp. 201–215.
6. B. Selman, H. Levesque, and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems," *Proc. 10th Nat'l Conf. Artificial Intelligence* (AAAI 92), AAAI Press, 1992, pp. 440–446.
7. D. Schuurmans and F. Southey, "Local Search Characteristics of Incomplete SAT Procedures," *Proc. 17th Nat'l Conf. Artificial Intelligence* (AAAI 00), AAAI Press, 2000, pp. 297–302.
8. H. Kautz and B. Selman, "Planning as Satisfiability," *Proc. 19th European Conf. Artificial Intelligence* (ECAI 92), Wiley, 1992, pp. 359–363.

9. R.E. Fikes and N. Nilsson, “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving,” *Artificial Intelligence*, vol. 5, no. 2, 1971, pp. 189–208.

10. H. Kautz and B. Selman, “Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search,” *Proc. 13th Nat’l Conf. Artificial Intelligence* (AAAI 96), AAAI Press, 1996, pp. 1194–1201.

11. A. Biere et al., “Symbolic Model Checking without BDDs,” *Proc. 5th Int’l Conf. Tools and Algorithms for the Analysis and Construction of Systems* (TACAS 99), LNCS 1579, Springer-Verlag,1999, pp. 193–207.

12. A. Tate, “Generating Project Networks,” *Proc. 5th Int’l Joint Conf. Artificial Intelligence* (IJCAI 77), Morgan Kaufmann, 1977, pp. 888–893.

13. J.S. Penberthy and D.S. Weld, “Temporal Planning with Continuous Change,” *Proc. 12th Nat’l Conf. Artificial Intelligence* (AAAI 94), AAAI Press, 1994, pp. 1010–1015.

14. L.A. Wolsey, *Integer Programming*, John Wiley, 1998.

15. T. Vossen et al., “Applying Integer Programming to AI Planning,” *Knowledge Eng. Rev.*, vol. 16, no. 1, 2001, pp. 85–100.

16. H. Kautz and B. Selman, “Blackbox: A New Approach to the Application of Theorem Proving to Problem Solving,” *Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search*, 1998, pp. 58–60.

17. S.A. Wolfman and D.S. Weld, “The LPSAT Engine & Its Application to Resource Planning,” *Proc. 16th Int’l Joint Conf. Artificial Intelligence* (IJCAI 99), 1999, pp. 310–316.

18. J. Bisschop and A. Meeraus, “On the Development of a General Algebraic Modeling System in a Strategic Planning Environment,” *Mathematical Programming Study*, no. 20, 1982, pp. 1–29.

19. R. Fourer, “Modeling Languages versus Matrix Generators for Linear Programming,” *ACM Trans. Mathematical Software*, no. 9, 1983, pp. 143–183.

20. C.A.C. Kuip, “Algebraic Languages for Mathematical Programming,” *European J. Operational Research*, vol. 67, no. 1, 1993, pp. 25–51.

21. R. Fourer, D.M. Gay, and B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed., Thomson Brooks/Cole, 2003.

22. M. Milano et al., “The Benefits of Global Constraints for the Integration of Constraint Programming and Integer Programming,” *Working Notes AAAI-2000 Workshop Integration of AI and OR Techniques for Combinatorial Optimization*, 2000.

23. J. Hooker et al., “A Scheme for Unifying Optimization and Constraint Satisfaction Methods,” *Knowledge Eng. Rev.*, vol. 15, no. 1, 2000, pp. 11–30.

24. P. Refalo, “Tight Cooperation and Its Application in Piecewise Linear Optimization,” *Proc. 5th Int’l Conf. Principles and Practice of Constraint Programming* (CP 99), 1999, pp. 375–389.

25. M.J. Stefik, “Planning with Constraints (MOLGEN: Part 1),” *Artificial Intelligence*, vol. 16, no. 2, 1981, pp. 111–140.

26. D. Joslin, *Passive and Active Decision Postponement in Plan Generation*, doctoral dissertation, Univ. of Pittsburgh, 1996.

27. J.M. Lever and B. Richards, “parcPLAN: A Planning Architecture with Parallel Action, Resources and Constraints,” *Proc. 9th Int’l Symp. Methodologies for Intelligent Systems* (ISMIS 94), 1994, pp. 213–222.

28. V. Liatsos, *Scaleability in Planning with Limited Resources*, doctoral dissertation, IC-Parc, Imperial College, 2000.

29. P. Van Beek and X. Chen, “CPlan: A Constraint Programming Approach to Planning,” *Proc. 16th Nat’l Conf. Artificial Intelligence* (AAAI 99), AAAI Press, 1999, pp. 585–590.

30. J. Rintanen and H. Jungholt, “Numeric State Variables in Constraint-Based Planning,” *Proc. 5th European Conf. Planning* (ECP 99), 1999, pp. 109–121.

31. B. Do and S. Kambhampati, “Solving Planning Graph by Compiling it into a CSP,” *Proc. 5th Int’l Conf. Artificial Intelligence Planning and Scheduling* (AIPS 2000), 2000, pp. 82–91.

32. É. Jacopin and J. Penon, “On the Path from Classical Planning to Arithmetic Constraint Satisfaction,” *Papers from the AAAI-2000 Workshop on Constraints and AI Planning*, tech. report WS-00-02, AAAI Press, 2000, pp. 18–24.

33. R.P. Goldman et al., “MACBeth: A Multi-Agent Constraint-Based Planner,” *Papers from the AAAI-2000 Workshop on Constraints and AI Planning*, tech. report WS-00-02, AAAI Press, 2000, pp. 11–17.

34. A. Nareyek, *Constraint-Based Agents—An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds*, LNAI 2062, Springer-Verlag, 2001.

35. J. Frank and A. Jónsson, “Constraint-Based Attribute and Interval Planning,” *Constraints*, vol. 8, no. 4, 2003, pp. 339–364.

36. N. Hyafil and F. Bacchus, “Conformant Probabilistic Planning via CSPs,” *Proc. 13th Int’l Conf. Automated Planning and Scheduling* (ICAPS 03), 2003, pp. 205–214.

37. A. Lopez and F. Bacchus, “Generalizing GraphPlan by Formulating Planning as a CSP,” *Proc. 18th Int’l Joint Conf. Artificial Intelligence* (IJCAI 03), 2003, pp. 954–960.

38. K. Currie and A. Tate, “O-Plan: The Open Planning Architecture,” *Artificial Intelligence*, vol. 52, no. 1, 1991, pp. 49–86.

39. A. Tate, B. Drabble, and R. Kirby, “O-Plan2: An Open Architecture for Command, Planning, and Control,” *Intelligent Scheduling*, M. Zweben and M.S. Fox, eds., Morgan Kaufmann, 1994, pp. 213–239.

40. A. Tate et al., “Using AI Planning Technology for Army Small Unit Operations,” *Poster Paper in Proc. 5th Int’l Conf. Artificial Intelligence Planning and Scheduling* (AIPS 2000), 2000, pp. 379–386.

41. P. Laborie and M. Ghallab, “Planning with Sharable Resource Constraints,” *Proc. 14th Int’l Joint Conf. Artificial Intelligence* (IJCAI 95), 1995, pp. 1643–1649.

42. N. Muscettola, “HSTS: Integrating Planning and Scheduling,” *Intelligent Scheduling*, M. Zweben and M.S. Fox, eds., Morgan Kaufmann, 1994, pp. 169–212.

43. K. Erol, J. Hendler, and D.S. Nau, “UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning,” *Proc. 2nd Int’l Conf. AI Planning Systems* (AIPS 94), 1994, pp. 249–254.

44. M. Paolucci, O. Shehory, and K. Sycara, “Interleaving Planning and Execution in a Multiagent Team Planning Environment,” *Electronic Trans. Artificial Intelligence*, section A, no. 4, 2000, pp. 23–43; www.ep.liu.se/ej/etai/2000/003.

45. K. Erol, J. Hendler, and D.S. Nau, “HTN Planning: Complexity and Expressivity,” *Proc. 12th Nat’l Conf. Artificial Intelligence* (AAAI 94), AAAI Press, 1994, pp. 1123–1128.

46. E.D. Sacerdoti, “The Nonlinear Nature of Plans,” *Proc. 4th Int’l Joint Conf. Artificial Intelligence* (IJCAI 75), Morgan Kaufmann, 1975, pp. 206–214.

47. D. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, 1988.

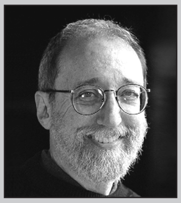
48. S. Mittal and B. Falkenhainer, “Dynamic Constraint Satisfaction Problems,” *Proc. 8th Nat’l Conf. Artificial Intelligence* (AAAI 90), AAAI Press, 1990, pp. 25–32.

49. P. Van Hentenryck, *The OPL Optimization Programming Language*, MIT Press, 1999.

50. P. Laborie, “Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results,” *Proc. 6th European Conf. Planning* (ECP 01), 2001.

The Authors

Alexander Nareyek is responsible for matters regarding artificial intelligence for the International Game Developers Association. He's also chairperson of the IGDA's Artificial Intelligence Interface Standards Committee. His main research interests include the enhancement of planning and constraint programming techniques with real-world features such as real time and dynamics. His primary application area is computer games. He received his PhD from the Technical University of Berlin. Contact him at alex@ai-center.com; www.ai-center.com/home/alex.



Eugene C. Freuder is a research professor and a Science Foundation Ireland Fellow at University College Cork, where he's the director of the Cork Constraint Computation Centre. His research interests include constraint-based reasoning and its applications. He received his PhD in computer science from MIT. He's a fellow of the AAAI and of the European Coordinating Committee for Artificial Intelligence. Contact him at the Cork Constraint Computation Centre, Univ. College Cork, Cork, Ireland; e.freuder@4c.ucc.ie.

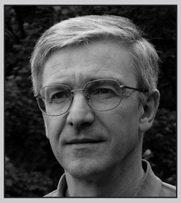
Robert Fourer is a professor of industrial engineering and management sciences at Northwestern University and is codirector of the Northwestern/Argonne Optimization Technology Center. His research focuses on large-scale optimization and the design of software tools for optimization, including the modeling language AMPL and the NEOS Server. Contact him at the Dept. of Industrial Eng. and Management Sciences, Northwestern Univ., 2145 Sheridan Rd., Room C210, Evanston, IL 60208-3119; 4er@iems.northwestern.edu.

Enrico Giunchiglia is an associate professor in the Department of Communication, Computer and System Sciences of the University of Genova. His main research interests are knowledge representation, planning, formal verification, and automated reasoning. He received his PhD in computer engineering from the University of Genova. He's a member of the AAAI and Artificial Intelligence Applications Institute. Contact him at DIST, Università di Genova, viale Causa 13, 16145 Genova (GE), Italy; giunchiglia@unige.it.

Robert P. Goldman is a senior scientist at Smart Information Flow Technologies, LLC. His research focuses on planning, particularly the boundary between planning and control theory, and on reasoning and action under uncertainty. He has worked on the constraint-based planner Macbeth and the CIRCA system for real-time controller synthesis. He received his PhD in computer science from Brown University. He's a member of the IEEE and AAAI. Contact him at Smart Information Flow Technologies, 211 N. First St., Suite 300, Minneapolis, MN 55401; rpgoldman@sift.info.

Henry Kautz is an associate professor in the Department of Computer Science and Engineering at the University of Washington. He's a fellow of the AAAI. Contact him at the Dept. of Computer Science & Eng., Sieg Hall, Room 417, Box 352350, Univ. of Washington, Seattle, WA 98195-2350; kautz@cs.washington.edu.

Jussi Rintanen is a research assistant and assistant professor at the Albert-Ludwigs-University Freiburg. His research interests include automated reasoning and planning, especially reasoning in propositional logics and logic-based approaches to different forms of planning, ranging from the classical deterministic planning problem to planning under partial observability. He received his PhD from the Helsinki University of Technology. Contact him at Albert-Ludwigs-Universität Freiburg, Institut für Informatik, AG KI, Georges-Köhler-Allee, 79110, Freiburg im Breisgau, Germany; rintanen@informatik.uni-freiburg.de.



Austin Tate is the technical director of the Artificial Intelligence Applications Institute and holds the Personal Chair of Knowledge-Based Systems at the University of Edinburgh. Contact him at AIAI, University of Edinburgh, Appleton Tower, Crichton St., Edinburgh EH8 9LE, UK; a.tate@ed.ac.uk.

51. H.J. Levesque etl al, "GOLOG: A Logic Programming Language for Dynamic Domains," *J. Logic Programming*, vol. 31, nos. 1–3, 1997, pp. 59–83.

52. C. Boutilier et al., "Decision-Theoretic, High-Level Agent Programming in the Situation Calculus," *Proc. 17th Nat'l Conf. Artificial Intelligence* (AAAI 00), AAAI Press, 2000, pp. 355–362.

53. A.L. Blum and M.L. Furst, "Fast Planning Through Planning Graph Analysis," *Artificial Intelligence*, vol. 90, nos. 1–2, 1997, pp. 281–300.

54. M.L. Ginsberg, "Do Computers Need Common Sense?" *Proc. 5th Int'l Conf. Principles of Knowledge Representation and Reasoning* (KR 96), 1996, pp. 620–626.

55. G. Verfaillie and T. Schiex, "Solution Reuse in Dynamic Constraint Satisfaction Problems," *Proc. 12th Nat'l Conf. Artificial Intelligence* (AAAI 94), AAAI Press, 1994, pp. 307–312.

56. M.H. Sqalli and E.C. Freuder, "Inference-Based Constraint Satisfaction Supports Explanation," *Proc. 13th Nat'l Conf. Artificial Intelligence* (AAAI 96), AAAI Press, 1996, pp. 318–325.

57. E.C. Freuder and R.J. Wallace, "Partial Constraint Satisfaction," *Artificial Intelligence*, vol. 58, nos. 1–3, 1992, pp. 21–70.

58. Z. Ruttkay, "Fuzzy Constraint Satisfaction," *Proc. 3rd IEEE Int'l Conf. Fuzzy Systems*, 1994, pp. 1263–1268.

59. S. Bistarelli et al., "Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison," *Constraints*, vol. 4, no. 3, 1999, pp. 199–240.

60. T. Walsh, "Stochastic Constraint Programming," *Proc. 15th European Conf. Artificial Intelligence* (ECAI 02), 2002, pp. 111–115.

61. M. Yokoo, *Distributed Constraint Satisfaction—Foundations of Cooperation in Multiagent Systems*, Springer-Verlag, 2001.