Beyond the Plan-Length Criterion

Alexander Nareyek

GMD FIRST, Kekuléstr. 7, D - 12489 Berlin, Germany
alex@ai-center.com
http://www.ai-center.com/home/alex/

Abstract. Resources such as food, hit points or magical power are standard features of today's computer games. Computer-guided characters often have goals that are related to these resources and must take this into account in planning their behavior. In the field of artificial intelligence, resource-based action planning systems have recently begun to mushroom. However, these systems still tend to neglect resources. Conventional plan length is the primary optimization goal, resources being of only secondary importance. This paper illustrates the problem and gives an overview of the Excalibur agent's planning system, which overcomes these deficiencies by applying an extended constraint programming framework to planning. It allows the search space to be explored without the need to focus on plan length, makes it possible to optimize other criteria like resource-related properties and promotes the inclusion of domain-dependent knowledge to guide and accelerate the search for a plan.

1 Introduction

A key factor in today's computer games is the computation of an appropriate behavior of computer-guided characters/mobiles/items (henceforth called *agents*). Reactive capabilities can be easily integrated by means of simple hard-wired if-then rules. Higher-level reasoning is hardly ever implemented, however. This is not because such methods are not needed. The reason is that there are hardly any techniques available because the field of artificial intelligence (AI) has focused mainly on simplified problems and algorithms with provable properties, such as completeness and termination. However, problems of optimizing an agent's behavior with respect to resource-related properties, such as the maximization of an agent's strength, belong to the complexity class of undecidable problems and are hardly ever addressed.

The AI field that deals with determining the right actions to pursue an agent's goals is *AI planning*[1]. The basic planning problem is 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

[1] Please note that the use of the term *planning* in AI is different from what people in the operations research (OR) community would expect (e.g., *scheduling*).

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.

Nearly all real-world (virtual-world) problems involve numerical resources, which restrict a planning system's decision options and may be subject to optimization criteria. In recent years, new planning systems have been developed that are able to handle resources. However, in most cases the resources are only used as auxiliary conditions to guarantee a valid plan. Even in most approaches that include an optimization of resource-related properties, the plan length still serves as the primary optimization criterion.

Sections 2 and 3 introduce the Orc Quest example and use it to show the inappropriateness of today's planning systems for resource-optimization problems. Section 4 presents an overview of the EXCALIBUR agent's planning system, which tackles these problems and explains the system's approach using the Orc Quest example. Key techniques are the use of an extended constraint programming framework and the application of local search guided by domain knowledge. Related work and conclusion are given in Section 5.

2 The Orc Quest Example

Imagine you are a computer-guided Orc in a computer game[2]. Your interests are somewhat different from those of your philistine Orc friends. To get them a little more acquainted with the fine arts, you decided to surprise them with a ballet performance. As a highlight, five of those delicate humans are to play supporting roles. However, the uncivilized humans do not appreciate the charm of Orc-produced art and are extremely reluctant to participate. This makes it rather painful to catch a human and force him to take part in the performance, and much more painful to catch a whole group of them. On the other hand, going for a group saves time. This can be formalized as follows:

```
STATEVARS: Duration, Pain, Performers ∈ ℕ0

INIT: Duration = 0, Pain = 0, Performers = 0
GOAL: Performers ≥ 5

ACTION catch_only_one:
 Duration += 2, Pain += 1, Performers += 1

ACTION catch_a_group:
 Duration += 5, Pain += 4, Performers += 3
```

[2] For those who have not heard of Orcs before: "Orcs ... are the most famous of Tolkien's creatures. ... Orcs tend to be short, squat and bow-legged, with long arms, dark faces, squinty eyes, and long fangs. ... Orcs hate all things of beauty and love to kill and destroy." [Mattsson 2000]

2.1 The Application of Planning Systems

Being an Orc, you lack sufficient brain power to solve the problem. You therefore apply a conventional AI planning system and come up with the following plan: `catch_a_group & catch_a_group`, which yields a plan duration of **10 hours**, an **8 on the pain scale** and **6 performers**. The plan attains your goal of catching at least 5 performers, but the other results look to you to be capable of improvement.

You realize that you have forgotten to apply an optimization criterion. You thus repeat the planning process, applying a state-of-the-art resource-based planning system that you request to minimize your pain, hoping dearly that none of your Orc friends will realize what a coward you are.

`GOAL: Performers` $\geq$ `5, MIN(Pain)`

Strangely enough, the resource-based planning system delivers the same plan as the conventional planning system.

2.2 The Real Optimum

Shortly afterward, the first group of humans is caught and you decide to get them to verify the planning system's results. The three gawky simpletons believe your promise to release them if they help you. After a while, they present you with a plan of five sequential `catch_only_one` actions, which yields a plan duration of **10 hours**, a **5 on the pain scale** and **5 performers**. This answer worries you as it involves much less pain than the solution found by the planning system employed to minimize your pain. Is human AI technology merely a way of undermining the Orcs' preordained dominance in the fine arts?

2.3 A Question of Relevance

You call the humans to account for the behavior of the planning system. They reply that most resource-based planning systems consider only a limited number of actions, increasing this number only if they are unable to produce a *correct plan*, the primary optimization goal being *plan length*. If it is possible to produce a correct plan with a certain number of actions, optimization of other goal properties can begin and plans with a larger number of actions are no longer considered. For your planning problem, the plan `catch_a_group & catch_a_group` is the only correct plan with a *minimal number of two actions*, and thus also the optimum with respect to any secondary optimization criterion.

Taking the number of actions or the plan length as the primary optimization criterion seems a very academic approach to you and you wonder if it has any relevance. You notice that the humans are becoming increasingly uncomfortable when they start arguing that optimization of the plan's duration is what is needed in most cases. And the duration would be the same as the number of actions. You no longer trust these humans and demand that they draw a picture of the complete search space. The result does not improve the humans' situation because the plan with the shortest duration is different from the one involving the minimal number of actions (see Figure 1).

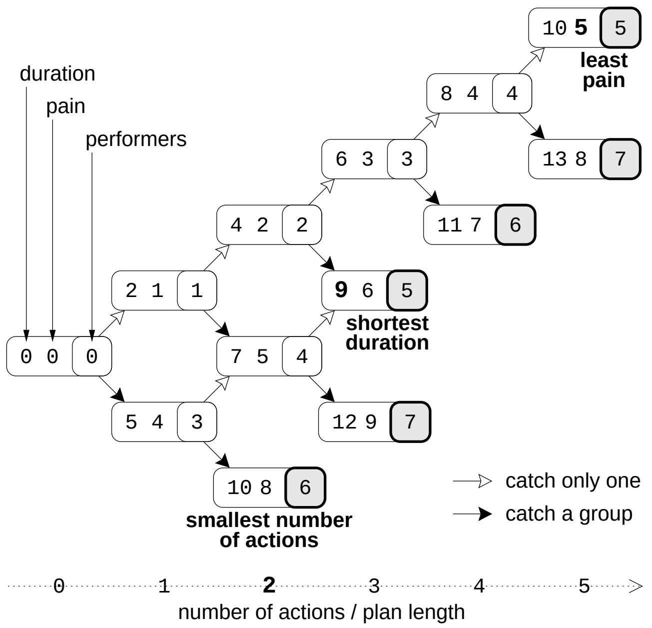

Fig. 1. The Search Space of the Orc Quest Example

Luckily for the group of humans, your goal is a plan involving the minimal amount of pain, and you reluctantly release the humans as promised because the plan involving the minimal amount of pain does not entail catching a whole group of humans. Somehow you feel dissatisfied at not having broken your promises.

3 The Nearby Wizard

A little bit later, it occurs to you that there is a wizard living nearby who can prepare a magic potion that lessens the pain by one on the pain scale. The drawback here is that the wizard is also very fond of humans because he is in continual need of test persons to improve his miserable skills in casting teleportation spells. Usually, then, he wants some humans in exchange for his magic potion, which he calls “ethanol”.

You visit the wizard and explain your problem to him. He scratches his head, walks around for a while deep in thought and finally says: “All right, this is what we’ll do: I’ll give you 11 phials of potion for every 10 humans that you promise to bring to me.” You are not really sure how many phials you should take and formalize this for your planning system as follows:

```
ACTION deliver_humans:
 Duration += 1, Pain -= 11, Performers -= 10
```

3.1 Plan-Length Bounds

The optimization problem without the `deliver_humans` action could have been optimally solved by calculating an upper bound for the plan length and starting the planning system with the requirement to consider only plans with a length of this bound. A `no_action` action could be introduced to allow for plans that are shorter than the length allowed by the bound. The bound is easy to calculate because a plan's `Performers` and `Pain` increase strictly monotonically with the plan length, and after an application of 5 actions the goal condition for the `Performers` is met anyway, and the `Pain` can only get worse for longer plans.

However, this simple way of calculating an upper bound is no longer applicable with an action like `deliver_humans` that destroys the monotony. You think about the new problem for a bit, but it is far too complex for an Orc. Lacking a method to calculate the upper bound, you decide to run the planning system with some random bounds for the plan length. Starting with a bound of 6, the plan that consists of 5 times `catch_only_one` is returned. With a length of 7, you still get this solution. You decide to raise the bound to 15, and ... wait. After a long period, the solution is confirmed again. It seems to be the optimum, but you test a bound of 30 just to be sure, and ... wait. After a while, the planning system returns a different solution than before. Unfortunately, this solution is beyond your comprehension and you are gradually overcome by the unpleasant feeling that the suggested plan "`out of memory`" is just another feature of the system designed to mislead helpless Orcs.

The wizard is quite astonished when you tell him that his offer would not help you to improve your plan. You offer to demonstrate this with your planning system, but he wants to try his own. Shortly afterward, he presents you with a plan that entails no pain for you. The plan has a plan length of 60, and you wonder how his system was able to return a solution that quickly.

3.2 A Question of Efficiency

"The wrong approach taken by your planning system," the wizard starts to explain, "is due to its focusing on qualitative problems, like the ancient blocks-world domain. The only feature distinguishing solutions was the plan length, and so everyone wanted to optimize this. A systematic exploration of the search space along an increasing plan length is an obvious way of tackling this problem, and such approaches are still at the heart of today's planning systems, even if the problems now involve optimization criteria that do not change strictly monotonically with the plan length."

In response to your questioningly look, he continues: "Your planning system has a very strange way of exploring the search space." He draws a small figure on the ground (see Figure 2).

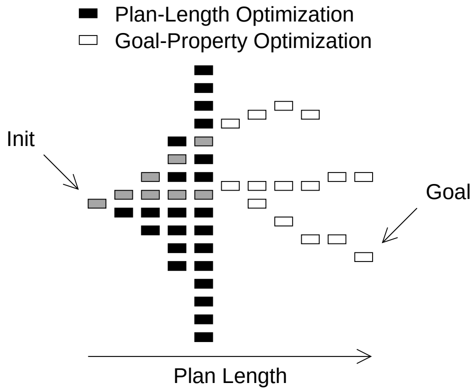

Fig. 2. Search-Space Exploration

"The plan-length optimization of your system," the wizard explains, "tries to verify that there is no valid solution for a certain plan length, and has to perform a complete search for this length before being able to increase the plan length. And even if you instruct it to begin with a certain length bound, it constructs the structures for the whole search space for this length, which is a pretty useless approach for non-toy problems.

As you have already realized, the plan-length criterion is a poor guide for the search because it has absolutely no relevance, so the search is rather like a random search. It is much more convenient to explore the search space without being bound to the plan length, as my planning system does. This allows us to perform a goal-property-driven search by integrating domain knowledge. Searching in an infinite search space means relying on powerful heuristics instead of being tied to a method of exploration that is determined by an irrelevant criterion."

3.3 Complexity

The wizard still sees a questioning look on the face in front of him, and he heaves a sigh: "Are you wondering about the infinite search space? Our planning problem is *undecidable* because it involves numbers with unpredictable bounds — you can read up on this in [Erol, Nau & Subrahmanian 1991]."

Suddenly, while casting a light-the-candle spell the wizard's apprentice vanishes in a big ball of fire. He had always been eager to try out this spell, but was warned by the wizard about how dangerous it was. Now the apprentice had been eavesdropping while you were talking with the wizard and had concluded that the optimization of his life's duration is also an undecidable problem, and thus not worth considering.

The wizard shakes his head sadly: "Hardcore theorists! If a problem is undecidable, they hate to tackle it." Just to be sure that you will not fall victim to one of the wizard's teleporation spells, you quickly put in: "I'm not one of them!" "No," the wizard smiles, "Orcs don't tend to be theorists anyway. But perhaps this neglect of undecidable problems is the reason why for so long no one dared to tackle problems other than plan-length optimization."

3.4 Decision-Theoretic Planning

The wizard invites you to dinner, and you gladly accept. A little while later, you are sitting in a comfortable armchair with a jar of the wizard's ethanol potion in your hands. The time has come to ask where he got his planning system from. The wizard's potion seems to have a paralyzing effect on your tongue, but finally you manage to put your question in reasonably comprehensible terms.

The wizard takes a while to answer and gives you an earnest look. Then he raises his eyebrows meaningfully: "Have you heard of EXCALIBUR?" You are seized with a shivering fit because EXCALIBUR is a popular name for swords used by computer players' avatars that generally enjoy tyrannizing harmless Orcs.

"No, no," the wizard says appeasingly, "not the sword!" His voice takes on a vibrant tone: "Did you ever consider the possibility that you are part of a computer program and that your behavior is largely guided by a planning system?" He pauses briefly to let you reflect on all this question's implications. "Well," he continues, casting a thunderbolt to conjure up a more threatening atmosphere, "what I mean is the EXCALIBUR Agent's Planning System!" He lowers his voice: "We are the *agents*. You understand?"

This sounds like one of those popular conspiracy theories. You are, however, not particularly well versed in philosophy and already have difficulty explaining why there are now two wizards and why all of the equipment in the room is swaying curiously. "Please put off describing the system for now," you say, putting an end to this rather dubious discussion. "If resources are such a vital component, why aren't there other planning systems like the EXCALIBUR agent's planning system?"

"Well," the wizards say, "I must admit that there have, of course, also been other approaches at trying to focus on optimizing things other than plan length. These are mostly subsumed under the term *decision-theoretic planning*. But most of the research in this area aims to optimize probabilistic issues for reasoning under conditions of uncertainty (see [Boutilier, Dean & Hanks 1999] for an overview). Although goals like maximizing probability to attain a goal can also be interpreted as resource-related optimization (the probability being the resource to be maximized), probability is usually considered to have only monotonical features, i.e., a plan cannot have a higher probability of attaining the goal than a subplan that also attains the goal. Your problem involving an action like using a magic potion has no counterpart here.

But there are also resource-oriented planning systems that are able to handle more complex problems, e.g., Williamson and Hanks' PYRRHUS planning system [Williamson & Hanks 1994] and Ephrati, Pollack and Milshtein's A*-based approach [Ephrati, Pollack & Milshtein 1996]. The difference between these and the EXCALIBUR agent's planning system is the conceptual approach."

"Sorry, did you say *A**? It's funny that you should mention this term because it sometimes comes mysteriously to mind — usually when I think about how to get to a certain place. Do you know what it means?"

"Well, all it means is that the EXCALIBUR agent's planning system has specialized heuristics for specific subtasks. Don't worry about it! But — A* brings me directly to the point. The resource-oriented planning systems mentioned above construct plans in a stepwise manner, e.g., action by action, and if they fail, they go back to an earlier partial plan and try to elaborate this. In order to choose a partial plan for an elaboration — called *refinement* — an upper bound of the quality is computed for each partial plan. The plan that yields the highest upper bound is chosen for elaboration. Sounds like a great concept, doesn't?"

"Probably not — since you ask!" you reply and applaud loudly, impressed by your own cleverness. The wizards continue: "Er, you're right, the concept is great for things like path finding, for which it is easy to determine bounds because the actions can only add values. But for complex planning problems it is not normally possible to calculate bounds on the quality value for a partial plan. We spoke about the problems with bounds before. Does a single action like building a trap for someone puts bounds on the total fun you will have?"

"Definitely not! Things can actually get worse!" you answer bad-temperedly, "If I don't take precautions, the person trapped may be a player's avatar, and then the world is usually reset to a former state only to build a trap for the person who will build the trap."

"Exactly," the wizards continue, "The EXCALIBUR agent's planning system takes a different approach. It iteratively repairs *complete grounded plans* for which a concrete quality can be determined. The repair methods can exploit the quality information to improve the plan, e.g., by replacing a set of actions that cause a loss in quality. This technique, based on local-search methods, is not so – let's say *unfocused* – with respect to the plan quality, and there is practically no restriction on the kind of objective function to be optimized (see also [ILOG, Inc. 2000]). The ASPEN system [Chien et al. 2000,Rabideau et al. 1999] is very similar to EXCALIBUR in this respect. To sum up, *decision-theoretic planning* is a very broad term, which, of course, also encompasses the EXCALIBUR agent's planning system."

4 The EXCALIBUR Agent's Planning System

"Let me give you an overview of the system", the wizards say. "Please consult the references that I'll give in the following for a more detailed description of the individual techniques.

4.1 Model Structures

In EXCALIBUR, a planning problem is formulated as a constraint satisfaction problem (CSP). A constraint is a test that checks a specific relation between certain variables. For example, the constraint EQUALS can check the equality between two variables. If all of a problem's constraints are fulfilled (or *satisfied* or *consistent*, terms also used here), the variable's values constitute a valid solution to the problem. A CSP specifies which constraints and variables exist and how the variables are related by which constraints. Where the planning *model* is mentioned in the following, this refers to the CSP used to specify the planning problem.

The planning model focuses on resources. A resource is a temporal projection of a specific property's state, which may be subject to constraints such as preconditions and changes. Numerical as well as symbolic properties are uniformly treated as resources. For example, a battery's POWER is a resource, but the PAIN you experience and a DOOR are, too:

POWER is [0: $t \in [0..5]$, $10 - 0.75 \times t$: $t \in [6..13]$, 0: $t \in [14..\infty[$],

PAIN is [0: $t \in [0..1]$, 1: $t \in [2..\infty[$],

DOOR is [OPEN: $t \in [0..45]$, CLOSED: $t \in [46..60]$, UNKNOWN: $t \in [61..\infty[$]."

"Why do you call them *resources* if they are merely state variables that can also have a numerical domain?" you ask. "The term *resource*," the wizards explain, "is a term commonly used in the constraint programming (CP) and OR community. If you prefer the terms *state variable* or *fluent*, then use these instead. For the planning model, we use the term *resources* because of the system's close relation to applications for resource allocation/optimization.

Figure 3 shows an example that informally illustrates the planning model's basic elements. Figure 4 gives an overview of the formal CSP representation; it does not show the CSP itself, but it does indicate possible relations between constraint types and variables for the planning CSP. The CSP itself can include multiple instances of the constraints and variables shown. Only some of the constraints are discussed here. For a more detailed description, please refer to [Nareyek 2000].

Oops, sorry, I forgot to tell you what the elements of Figure 4 mean. A variable is depicted by a circle, a constraint by a rectangle. Arrows are used to connect variables to constraints. The role/position of a variable within a constraint can be expressed by an arrow's label and direction.

In addition, there are special types of constraints — so-called *object constraints* — that do not check the variables' values, but provide structural context information. For example, it must be known which two *Begin* and *End* variables together form an ACTION TASK. Otherwise, an ACTION RESOURCE CONSTRAINT designed to check if the connected ACTION TASKs overlap might combine *Begin* and *End* variables of different ACTION TASKs for the check. Object constraints are represented by a rectangle with a dashed outline.

An action (e.g., eating a peanut) consists of a set of different preconditions that must be satisfied (e.g., that the agent has an peanut), operations that the agent has to perform (e.g., the mouth's operations) and resulting state changes (e.g., the agent's hunger being satisfied). These elements are represented by *tasks*, i.e., there are PRECONDITION TASKs for precondition tests, ACTION TASKs for operations and STATE TASKs for state changes. The tasks are collections of

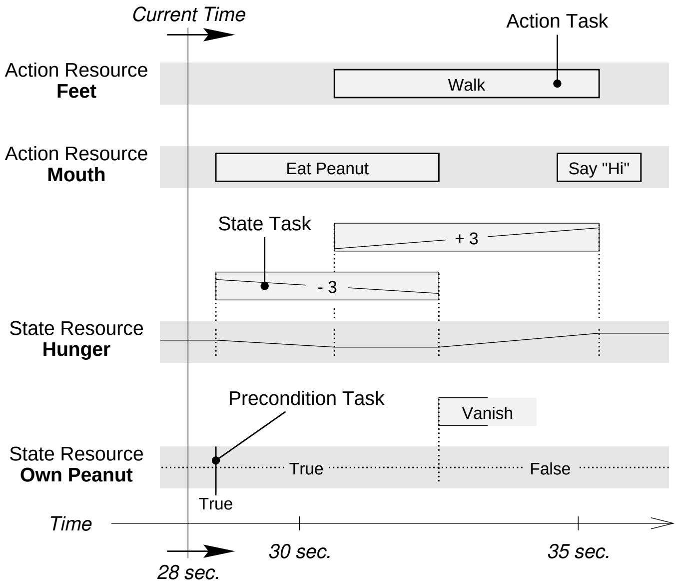

Fig. 3. Visualization of the Basic Plan Elements

variables, e.g., temporal variables that determine the beginning and end of the tasks, or variables that specify how a state's properties are changed by the action. For example, an action `catch_a_group` includes a State Task that has a *Contribution* variable with a value of 4, which represents the resulting increase in pain.

Action Resource Constraints ensure that the agent has enough capacity to carry out the operations that are needed to execute the plan's actions. The Action Resource Constraint correlates Action Tasks and ensures that the tasks do not overlap. For your problem, we need only a general Orc Action Resource Constraint, as there is no further refinement regarding your hands, feet, mouth or whatever. The Orc Action Resource Constraint can be linked to Action Tasks with a duration of 2 hours (if the task's *Operation* variable has the value `catch_only_one`), 5 hours (for a value of `catch_a_group`) and 1 hour (for a value of `deliver_humans`).

A State Resource Constraint temporally projects the state of a specific environment property and checks if the plan's related preconditions are satisfied. The constraint relates (via a State Resource) State Tasks that cause the change in the property's state and Precondition Tasks that specify the tests. For your problem, we need two State Resource Constraints, Pain and Performers, that support discrete numerical domains and integrate State Tasks by adding their pain/performers values. For example, the Pain State Resource Constraint's projection of your pain will be increased by 4 if a State Task that has a *Contribution* variable with a value of 4 is connected.

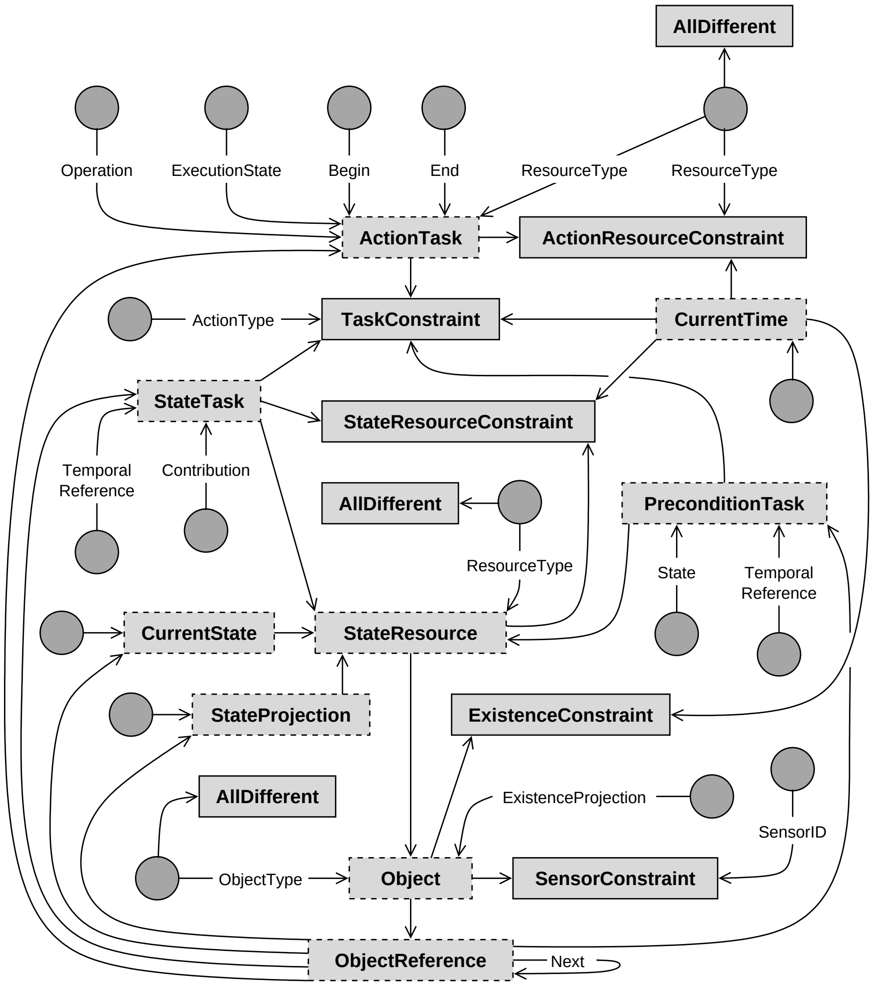

Fig. 4. Possible Type Relations

A Task Constraint represents an action by specifying a relation between Precondition Tasks, Action Tasks and/or State Tasks. It ensures that the connected tasks represent a valid action by enforcing a certain task configuration (i.e., the constraint specifies which tasks are to be related to the Task Constraint and what kind of restrictions apply to the tasks' variables). For your problem, four different configurations satisfy a Task Constraint. These express the three actions `catch_only_one`, `catch_a_group` and `deliver_humans`, each of them having an Action Task for the Orc Action Resource Constraint, a State Task for the Pain State Resource Constraint and a State Task for the Performers State Resource Constraint. A fourth configuration consists of a Precondition Task for the Performers State

RESOURCE CONSTRAINT to establish the satisfaction goal `Performers` $\geq$ `5`. In addition, the TASK CONSTRAINT requires that its tasks' variables have the appropriate values (e.g., if the TASK CONSTRAINT's *ActionType* variable has a value of `catch_a_group`, a value of 4 is required for the *Contribution* variable of the STATE TASK that is to be connected to the PAIN STATE RESOURCE CONSTRAINT).

Because it is not required for your example, I will not detail the role of constraints like OBJECTS, EXISTENCE CONSTRAINTS, SENSOR CONSTRAINTS, etc. These are needed to realize an open-world planning in which the type and number of the world's objects is not known in advance and can change without any action on the part of the agent.

You may have noticed that the constraints are not typical low-level constraints such as LESS or EQUALS. Indeed, a key aspect of the EXCALIBUR agent's planning system is the use of so-called *global constraints*. A global constraint is a substitute for a set of lower-level constraints, additional domain knowledge allowing the application of specialized data representations and algorithms to guide and accelerate search. Global constraints are very important for speeding up search, and the success of commercial constraint programming tools like the ILOG Scheduler [Le Pape 1994] can be ascribed mainly to their use of global constraints.

4.2 Structural Constraints

If constraint programming is to be applied to planning, we face the problem that conventional formulations for constraint satisfaction problems are too restrictive because all the elements and their relations have to be specified in advance, e.g., that exactly *five* ACTION TASKs would be related to the ORC ACTION RESOURCE CONSTRAINT. But possible plans can be structurally very different and require different underlying CSPs, e.g., CSPs with five, six or seven ACTION TASKs that are related to the ORC constraint. Of course, it is possible to create a maximal CSP that incorporates all possible structures as subproblems, which is similar to what most resource-based planning systems do with their plan-length bounds. But the problem here is that maximal structures scale *very* badly and are only feasible for small and only slightly variable structures, which is not the case with planning problems. And as we don't want to be restricted to plans of a certain length, this is not an option. Thus, an extension of CSP formulations must be used that includes the search problem for the CSP structure and is not tied to a special procedure like an extension along an increasing plan length.

A so-called *structural constraint satisfaction problem (SCSP)* can be used to overcome these deficiencies. In an SCSP, the constraint graph is not explicitly given. Only the types of constraints (together with their possible relations) and a set of *structural constraints* that restrict certain graph constellations are specified. A solution to the constraint problem is a constraint graph with an arbitrary number of variables, conventional (or object) constraints and connecting arrows, such that the graph and the variable assignments not only satisfy the conventional constraints (such as an ACTION RESOURCE CONSTRAINT) but also the structural constraints. The formalism is based on algebraic graph grammars [Rozenberg 1997].

Figure 5 shows an example of a structural constraint that checks if every Action Task is linked to an Action Resource Constraint with the same *ResourceType* variable. This prevents invalid Action Task assignments, e.g., the Action Task of walking being scheduled by the Action Resource Constraint of the mouth. If the left-hand side of a structural constraint matches the constraint graph at some point, one of its right-hand sides (only one in the figure) has to match, too. For more details on structural constraints and SCSPs, please refer to [Nareyek 1999a] and [Nareyek 1999b]. A complete description of the structural constraints required for planning can be found in [Nareyek 2000].

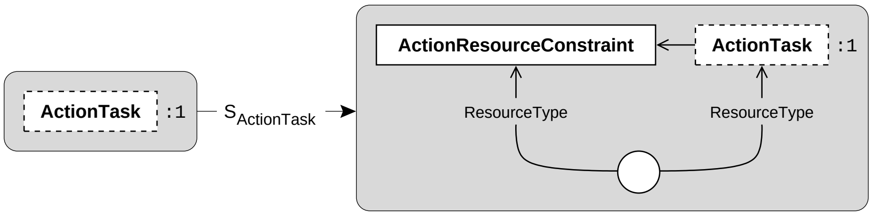

Fig. 5. The Structural Constraint Action Task

The SCSP formulation is used to automatically derive rules/productions that can be applied to change the constraint graph during search. Thus, the search for valid variable assignments and for a valid graph structure can be seamlessly integrated.

But we have only talked about the specification of planning problems. Now I'm going to tell you how local search is used to find a solution to the problem."

4.3 Local Search

"Local search?" you say rather disdainfully because you have read somewhere that local search is something quite unscientific. Luckily, the annoyed wizard realizes that you are at present the only person he has to talk to (the other wizard appears to have left, presumably to get more of the great ethanol potion) and abstains from casting a teleportation spell on you to remedy the situation. "As I already mentioned," he resumes, "we have an undecidable/infinite search space where we can neither find all solutions nor verify that no solution exists. So the advantages of complete refinement search can't be exploited. On the other hand, the fact that local search is an anytime algorithm and that the techniques are not normally affected by dynamic changes in the search space make local search the ideal candidate for the computer-games domain. We live in a highly dynamic world and must react in real time!"

To cover up your previous faux pas, you grunt in assent. The wizard gives you a rather disconcerted look, but quickly collects himself and continues: "Another advantage of local search is that it is inherently a partial constraint satisfaction [Freuder & Wallace 1992]. This means that it is possible to find reasonable plans even if the problem is over-constrained, i.e., if it is not possible to create a fully correct plan. An extension of refinement search to handle partial constraint satisfaction would be much more inefficient, as the search space grows vastly.

Let's move on to how local search is accomplished. Local search means that there is always a complete assignment for the variables, and this assignment is iteratively changed toward a satisfactory/optimal solution. The quality of a current solution is measured by a so-called *objective function*. In contrast to an off-line refinement search, the iterative repair/improvement steps of local search make it possible to easily interleave sensing, planning and execution. After each single repair step, which can usually be computed very quickly, the planning process can involve changes in the situation (see Figure 6).

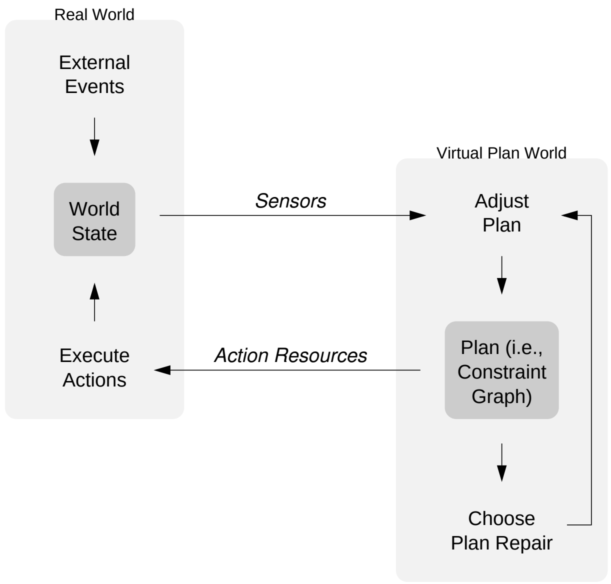

Fig. 6. Interleaving Sensing, Planning and Execution

The approach of [Nareyek 2001] is applied in the EXCALIBUR agent's planning system, which focuses the constraints to cause plan improvements. Each type of constraint can have a constraint-specific cost and goal function. Using the *cost function*, a constraint computes its inconsistency value, which expresses the degree of misassignment of the related variables. In connection with structural constraint satisfaction, invalid graph structures can also be penalized by the cost function (see [Nareyek 1999b]). Using the *goal function*, a constraint computes a preference value, which is used for optimization purposes.

For your problem, the PERFORMERS constraint's cost function is initialized to return the current plan's number of missing performers (to satisfy the PRECONDITION TASK of a TASK CONSTRAINT with an *ActionType* variable that represents the TASK CONSTRAINT's fourth configuration), the PAIN constraint's goal function to return the plan's resulting pain, and the ORC constraint's goal function to return the plan's total duration.

In addition, a constraint has *constraint-specific heuristics* that can change the values of the variables involved in order to reduce the constraint's inconsistency or improve its goal function value, e.g., changing a *Begin* variable to move a task to a better position in time. Domain-specific knowledge can be incorporated into the heuristics. In connection with structural constraint satisfaction, constraints can also use productions to change the graph in order to make an improvement, e.g., to add resources or actions. Thus, the search proceeds through a space consisting of complete but not necessarily optimal or correct plans.

4.4 The Constraints' Heuristics

Your problem does not require an application of sophisticated heuristics. The improvement heuristics can be the same for the PERFORMERS, PAIN and ORC constraint. To change the plan (i.e., the CSP), there are six modification alternatives[3]:

- add a `catch_only_one` TASK CONSTRAINT
- delete a `catch_only_one` TASK CONSTRAINT
- add a `catch_a_group` TASK CONSTRAINT
- delete a `catch_a_group` TASK CONSTRAINT
- add a `deliver_humans` TASK CONSTRAINT
- delete a `deliver_humans` TASK CONSTRAINT

For all six decision alternatives, there is a constraint-internal value that represents the preference for this alternative. All preference values are initially set to 1. If a constraint is called to effect an improvement of the plan, it increases an alternative's value by one if this alternative was chosen by the constraint's last improvement decision and the constraint's goal-/cost-function value is now better than the last time the constraint was called. If the goal-/cost-function value has deteriorated or is the same as last time, the preference value is decreased by two. Each time there is a consecutive deterioration, the decrease is doubled. However, no preference value can fall below one. Then, an alternative is chosen with a probability that is proportional to the alternative's preference value.

Let me illustrate the heuristic using the PAIN STATE RESOURCE CONSTRAINT. Consider a situation in which the current plan consists of 5 `catch_only_one` actions (C1A), 10 `catch_a_group` actions (CGA) and 3 `deliver_humans` actions (DHA) and the PAIN constraint is chosen to perform a plan improvement (see Situation 1 in Figure 7).

[3] A TASK CONSTRAINT is always added/deleted together with its required tasks.

	Current Plan			Pain State Resource Constraint							
				Goal Function Value	Preference Values						
	#C1A	#CGA	#DHA		+C1A	-C1A	+CGA	-CGA	+DHA	-DHA	Penalty
Situation 1	5	10	3	12	1	2	1	3	3	1	2
					1	2	1	3	4	1	2
					select +DHA						
	5	10	4								
...											
Situation 2	5	14	4	17	1	2	1	3	4	1	2
					1	2	1	3	2	1	4
					select +DHA						
	5	14	5								
...											
Situation 3	8	16	5	17	1	2	1	3	2	1	4
					1	2	1	3	1	1	8
					select -CGA						
	8	15	5								
...											
Situation 4	11	15	5	16	1	2	1	3	1	1	8
					1	2	1	4	1	1	2

Fig. 7. The Heuristic of the PAIN STATE RESOURCE CONSTRAINT

The PAIN constraint's goal function calculates the sum of the *Contribution* variables of the linked STATE TASKS. There are currently 5 linked STATE TASKS with a *Contribution* of 1, 10 with a *Contribution* of 4, and 3 with a *Contribution* of -11. Thus, the constraint's goal function has a value of $5 \times 1 + 10 \times 4 + 3 \times (-11) = 12$. The current preference values of the constraint are given in Figure 7 ("+" means addition and "-" means deletion).

Let's assume that the constraint's goal function value is now better than the last time the constraint was called and that the constraint's last improvement decision was to add a `deliver_humans` action. Accordingly, this decision's preference value is rewarded by increasing it by one (next line in Figure 7). Next, an alternative to improve the current situation is selected. The choice probability for the +DHA option is the highest (preference value divided by the sum of all preference values; $4/12 = 33\%$), and we assume that this alternative is chosen. The plan/CSP is changed according to this option.

After some iterations, the PAIN constraint can be called again (see Situation 2 in Figure 7). In the meantime, the plan has been changed and some more `catch_a_group` actions are added. The PAIN constraint's goal function value has deteriorated since the last time the constraint was called and its last decision's preference value is therefore decreased by the penalty value of 2. In addition, the penalty value is doubled. The choice probability for the +DHA option is now only $2/10 = 20\%$, but we assume that this alternative is chosen again. The plan is changed according to this option.

After some time, the PAIN constraint is called once more (Situation 3). The constraint's goal function value is the same as at the last call. Stagnation is considered to be as bad as deterioration, and the +DHA option's preference value is therefore decreased by the penalty value of 4, and the penalty value is doubled again. However, the +DHA preference value is increased to its minimum of 1 to ensure that the option retains a chance of being chosen. We assume that the -CGA alternative is chosen this time (probability of $3/9 = 33\%$). The plan is changed according to this option.

At the next call of the PAIN constraint (Situation 4), the constraint's goal function value has improved, and so, the -CGA preference value is increased and the penalty value is set back to 2.

4.5 The Global Search Control

As described above, global constraints have integrated heuristics to enable them to choose a successor state on their own. For each iteration of local search, a so-called *global search control* selects the constraint that is to perform the change. The choice depends on the constraints' cost and goal function values and implements the goal criteria.

The cost and goal functions may compete such that an improvement of one function value may lead to a deterioration in the other function value. Thus, various options for the selection of a function for improvement are customizable, e.g., alternating phases of a predefined length or with special abort criteria, or a general cost measure consisting of the addition of the single function values multiplied by static (or even variable) coefficients. A constraint that is selected for improvement is told whether to improve its goal or its cost function.

For your problem, within the global search control, we will always choose the PERFORMERS constraint to perform an improvement if the plan does not yield at least 5 performers. Otherwise, if the plan yields any pain, we'll choose the PAIN constraint. If we have a plan with enough performers and no pain, we'll choose the ORC constraint to help shorten the plan. Thereby, we can realize a search that has the performers criterion as satisfaction goal, the pain criterion as the primary optimization goal and the duration criterion as the secondary optimization goal.

The global search control also serves as a general manager, which variables and constraints can dock on to and off from in a plug-and-play manner. This is necessary if the constraint graph is changed by heuristics that make structural changes and it provides a simple mechanism for tackling problems that involve dynamic changes.

We'll initialize the search process with a CSP that includes the PERFORMERS, PAIN and ORC constraint and a TASK CONSTRAINT with an *ActionType* variable that represents the TASK CONSTRAINT's fourth configuration (to establish the satisfaction goal `Performers` $\geq$ `5`). The STATE RESOURCE CONSTRAINTS PERFORMERS and PAIN are initialized with *CurrentState* values of 0.

During search, the structural constraints have to be enforced in the same way as the other constraints. But if one can prove that the potential changes in the constraints' heuristics cannot violate any structural constraints, this can be skipped. For example, the additions/deletions of actions in our constraints' decision alternatives can easily be performed in a way that do not threaten structural constraints.

4.6 Solving the Orc Quest Problem

If you now start the planning system, you will quickly obtain the shortest possible plan that results in no pain for you (55 times `catch_only_one` and 5 times `deliver_humans`, which yields a plan duration of **115 hours**, a **0 on the pain scale** and **5 performers**). Figure 8 shows the temporal distribution for 100,000 test runs with different random seeds (e.g., after 1,000 improvement iterations, 100 % of the test runs found a plan that yields enough performers, 94 % of the test runs found a plan that yields enough performers and no pain, and 41 % of the test runs found the shortest plan that yields enough performers and no pain)."

"That looks nice," you comment, "but what about a comparison with other planning systems?" The wizard shakes his head: "Don't you remember the problems you had? The other systems will either come up with the plan that primarily minimizes the plan length (`catch_a_group & catch_a_group`) or require a bound for the plan length. If we cheat in favor of the other planning systems and provide the information that the optimal length bound is 60 actions, an encoding according to CPlan [Van Beek & Chen 1999] of this simple problem

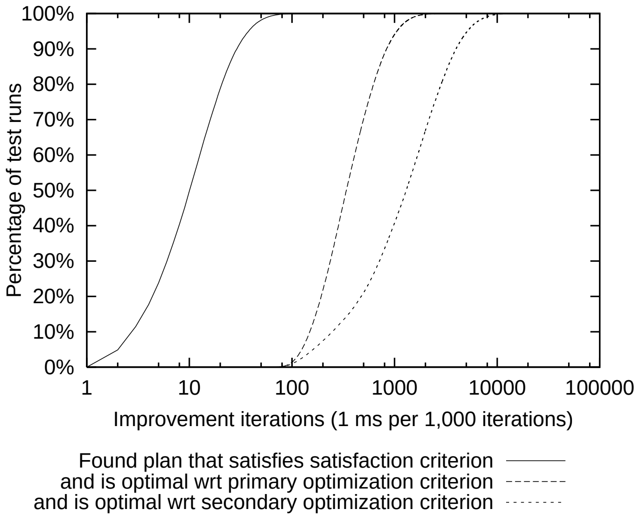

Fig. 8. Test-Run Results for the Orc Quest Example

would require 129,600 variables[4] — not to speak of constraints! Even the time required to create a list of these domain variables (tested with CHIP, which is a state-of-the-art constraint programming system) is longer than any of the test runs in Figure 8 needed to compute the optimal plan. However, we actually cheated in favor of CPlan by providing the optimal length bound. In general, the comparison is not a quantitative question of speed but a qualitative question of *capability* to find the optimum.

However, if you insist on quantitative results, we can start to improve the constraints' heuristics. As I said, domain-dependent knowledge can easily be integrated. The heuristics are quite general so far and don't make much use of your specific problem's characteristics. For example, we could improve the Pain constraint by paying more attention keeping the same number of performers if an option to reduce the pain is chosen. For this purpose, three additional `catch_only_one` actions can be added if the option to delete a `catch_a_group` action is chosen. In the same way, the number of performers can be replenished by adding `catch_only_one` and `catch_a_group` actions (partitioning them according to the preference values) if the option of adding a `deliver_humans` action is chosen. Even such simple modifications make it possible to reduce execution times by 20 % (compared to the results in Figure 8). However, the selection mechanisms involved can also be improved. For example, if the constraint-internal choice of a modification option is changed, such that the option with the highest preference value is chosen (instead of making a choice based on a probabilistic distribution), a further reduction of 20 % can be achieved.

[4] $\mathbf{60}$ steps $\times$ (duration domain $\mathbf{60} \times \mathbf{6}$ + pain domain $\mathbf{60} \times \mathbf{16}$ + performers domain $\mathbf{60} \times \mathbf{14}) = \mathbf{129{,}600}$.

5 Related Work and Conclusion

I have explained the approach of the EXCALIBUR agent's planning system. Your problem was a very simple one but it allowed me to outline the system's basic features and show why existing approaches are not able to handle it adequately. Specialized ACTION and STATE RESOURCE CONSTRAINTS were applied to your problem — of course, for more sophisticated problems involving precondition requirements and temporal aspects, more general versions must be available. More general / domain-independent versions of the planning model's constraints have already been developed and successfully tested on problems like the Logistics Domain. The constraints' heuristics also include operations that take into account the temporal distribution, like temporally shifting tasks to more suitable time points. Figure 9 shows run-time distributions for some simple problems of the AIPS-2000 planning competition (Track 2). I will report on this in more detail soon.

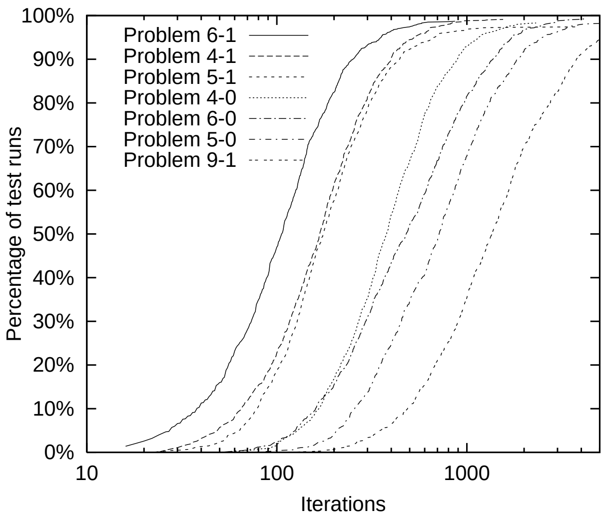

Fig. 9. Run-Time Distributions for the Logistics Domain

For planning problems that involve resource constraints, some planning systems keep planning and resource allocation separate, e.g., the approach of [Srivastava & Kambhampati 1999] and the *parc*PLAN system by [El-Kholy & Richards 1996]. The separation of planning and resource allocation prevents the systems from considering interactions between the decisions with respect to planning and resource assignment, which is a great disadvantage if resource-related properties are to be optimized. Consequently, the resources serve only as constraints for the planning problem and are not used as primary optimization goals. The same applies to resource-based planning systems that integrate planning and resource allocation, like O-Plan2 [Drabble & Tate 1994], IxTeT [Laborie & Ghallab 1995], the LPSAT engine's application to planning [Wolfman & Weld 1999] and IPP's extension [Koehler 1998]. All of these focus primarily on optimization of the plan length, which is a rather curious approach as this property is usually irrelevant.

Resource-based planning systems that are based entirely on general search frameworks like CP or OR require bounds for the plan length or the number of actions. If a correct plan cannot be found, these bounds can be expanded. Some examples here are the OR-based approach of [Bockmayr & Dimopoulos 1998], ILP-PLAN [Kautz & Walser 1999], CPlan [Van Beek & Chen 1999] and the approach of [Rintanen & Jungholt 1999]. Again, these systems primarily optimize the plan-length property. An optimum with respect to resource-related optimization goals can only be found if the initial bound can be set such that the optimal solution is guaranteed to lie within this bound. This is a very hard task for specific problems and impossible at a general level. Besides, creating the maximal structures for the search space is much too costly for complex real-world problems."

"What do you mean by *real world*?" you put in doubtfully. The wizard quickly corrects himself: "Sorry, I meant of course: for complex virtual-world problems.

Most planning systems use highly specific representations and algorithms, which makes it very hard to integrate new features like reasoning about resources. Unlike these planning formalisms, the EXCALIBUR agent's planning system is based on the general search framework of constraint programming. In contrast to other general search frameworks such as propositional satisfiability (SAT) and OR (i.e., integer linear programming), the CP framework allows the use of higher-level constraints. The propositional clauses of SAT and the linear inequations of OR can't exploit higher-level domain knowledge to support search. Besides, the SAT approach doesn't have numbers in its representation repertoire. Another reason for not using OR 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.

The concept of using global constraints for local search was applied, which enables us to integrate domain-dependent knowledge using the global constraints in a plug-and-play manner. As the search space is undecidable, this heuristic knowledge is essential to guide and accelerate search. Other local search approaches to planning, like SATPLAN [Kautz & Selman 1996], GPG [Gerevini & Serina 1999] and LPSP [Brafman & Hoos 1999], do not use domain-dependent knowledge. The closest relative to the EXCALIBUR agent's planning is the ASPEN system [Chien et al. 2000,Rabideau et al. 1999][5]. This system uses local search for resource-restricted planning problems in a similar way. However, the sensing and handling of an open world is somewhat limited and the system is not based on a general search framework like constraint programming.

To make the search independent of the focus on plan length, the paradigm of structural constraint satisfaction was used. This extension allows the search

[5] See also other articles in this book.

to make all possible kinds of changes in the constraint graph during the search process — not only after the current graph has been completely searched, and not only along an increasing plan length. Unlike other planning systems that use productions/rules to change the plan as part of the search (e.g., [Ambite & Knoblock 1997]), modeling planning as an SCSP allows us to specify the problem in a declarative manner and enables the corresponding productions to be deduced automatically. The automatic method guarantees that local-search methods can potentially find all valid plans, which is not normally the case with manual solutions.

The planning model of the EXCALIBUR agent's planning system borrows from typical constraint-based applications for resource allocation/optimization. The power of global constraints for constraint-specific representation and reasoning mechanisms for specific resource types was recognized here very early on and led to significant speedups in the solution process. General frameworks for planning and scheduling like HSTS [Muscettola 1994] lack such specialized representation and reasoning capabilities.

In conclusion, we can say that the EXCALIBUR agent's planning system conducts the search in a way that focuses on the optimization of resource-related properties. Even though the model supports full domain-independent planning, domain-specific heuristic knowledge can easily be integrated. The local search approach enables the system to provide very primitive plans (reactions) for short-term computation horizons, while longer computation times are used to improve and optimize the agent's behavior. The system's properties make it perfectly suitable for the application domain of computer games. Not only genres like computer role-playing games and strategy games can benefit from this. For games like first-person shooters, too, increasing efforts are being made to include a sophisticated behavior for their bots."

The wizard appears to have no more to say, so you chip in with a comment designed to keep him talking until you have drained your jar: "I've heard of something called hierarchical planning." Obviously, you've struck it lucky because the wizard answers gleefully: "Now that's interesting! The hierarchical planning approach has some features in common with the EXCALIBUR agent's planning system. Hierarchical planning is not bound to the plan length either and is also a kind of heuristic search. But the refinement hierarchy is usually created manually and rarely has the potential to construct every possible solution. Actually, I'm confident that the EXCALIBUR agent's planning system will soon be extended by hierarchical planning mechanisms. The hierarchy would set a general heuristic direction, allowing the iterative repair to vary the plan as needed."

Just as you are about to leave the wizard to set off on your hunt for humans, you have the sudden inspiration that more information on the underlying EXCALIBUR project is available at:

`http://www.ai-center.com/projects/excalibur/`

You have no idea what brought this on or what it means and put it down to an undocumented subfunction of your behavior program, triggered by the approach of the acknowledgments.

6 Acknowledgments

The work reported here is supported by the German Research Foundation (DFG), NICOSIO, Conitec Datensysteme GmbH and Cross Platform Research Germany (CPR).

References

[Ambite & Knoblock 1997] Ambite, J. L., and Knoblock, C. A. 1997. Planning by Rewriting: Efficiently Generating High-Quality Plans. AAAI-97, 706–713.

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

[Bockmayr & Dimopoulos 1998] Bockmayr, A., and Dimopoulos, Y. 1998. Mixed Integer Programming Models for Planning Problems. In Working Notes of the CP98 Workshop on Constraint Problem Reformulation.

[Boutilier, Dean & Hanks 1999] Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11: 1–94.

[Brafman & Hoos 1999] Brafman, R. I., and Hoos, H. H. 1999. To Encode or not to Encode – I: Linear Planning. IJCAI-99, 988–993.

[Chien et al. 2000] Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN — Automating Space Mission Operations using Automated Planning and Scheduling. In Proceedings of the Sixth International Symposium on Technical Interchange for Space Mission Operations and Ground Data Systems (SpaceOps 2000).

[Drabble & Tate 1994] Drabble, B. and Tate, A. 1994. The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. AIPS-94, 243–248.

[El-Kholy & Richards 1996] El-Kholy, A., and Richards, B. 1996. Temporal and Resource Reasoning in Planning: the *parc*PLAN approach. ECAI-96, 614–618.

[Ephrati, Pollack & Milshtein 1996] Ephrati, E.; Pollack, M. E.; and Milshtein, M. 1996. A Cost-Directed Planner: Preliminary Report. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 1223–1228.

[Erol, Nau & Subrahmanian 1991] Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1991. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. Technical Report CS-TR-2797, University of Maryland, Institute for Advanced Computer Studies, Maryland, USA.

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

[Gerevini & Serina 1999] Gerevini, A., and Serina, I. 1999. Fast Planning through Greedy Action Graphs. AAAI-99, 503–510.

[ILOG, Inc. 2000] ILOG, Inc. 2000. Optimization Technology White Paper — A comparative study of optimization technologies. White Paper, ILOG, Inc., Mountain View, CA.

[Kautz & Selman 1996] Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. AAAI-96, 1194–1201.

[Kautz & Walser 1999] Kautz, H., and Walser, J. P. 1999. State-space Planning by Integer Optimization. AAAI-99, 526–533.

[Koehler 1998] Koehler, J. 1998. Planning under Resource Constraints. ECAI-98, 489–493.

[Laborie & Ghallab 1995] Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. IJCAI-95, 1643–1649.

[Le Pape 1994] Le Pape, C. 1994. Implementation of Resource Constraints in ILOG Schedule: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering* 3(2): 55–66.

[Mattsson 2000] Mattsson, C. 2000. The Tolkien Monster Encyclopedia. `http://home7.swipnet.se/~w-70531/Tolkien/`

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

[Nareyek 1999a] Nareyek, A. 1999. Structural Constraint Satisfaction. In Papers from the AAAI-99 Workshop on Configuration, 76–82. (available via the EXCALIBUR project's webpage)

[Nareyek 1999b] Nareyek, A. 1999. Applying Local Search to Structural Constraint Satisfaction. In Proceedings of the IJCAI-99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business. (available via the EXCALIBUR project's webpage)

[Nareyek 2000] Nareyek, A. 2000. Open World Planning as SCSP. In Papers from the AAAI-2000 Workshop on Constraints and AI Planning, Technical Report, WS-00-02, 35–46. AAAI Press, Menlo Park, California. (available via the EXCALIBUR project's webpage)

[Nareyek 2001] 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. (available via the EXCALIBUR project's webpage)

[Rabideau et al. 1999] Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS).

[Rintanen & Jungholt 1999] Rintanen, J., and Jungholt, H. 1999. Numeric State Variables in Constraint-based Planning. ECP-99.

[Rozenberg 1997] Rozenberg, G. ed. 1997. *The Handbook of Graph Grammars. Volume I: Foundations*. Reading, World Scientific.

[Srivastava & Kambhampati 1999] Srivastava, B., and Kambhampati, S. 1999. Scaling up Planning by teasing out Resource Scheduling. ECP-99.

[Van Beek & Chen 1999] Van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. AAAI-99, 585–590.

[Williamson & Hanks 1994] Williamson, M., and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 176–181.

[Wolfman & Weld 1999] 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.