Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Configuration Planning with Temporal Constraints

Uwe Köckemann and **Lars Karlsson**
Center for Applied Autonomous Sensor Systems, Örebro University, Sweden

Abstract

Configuration planning is a form of task planning that takes into consideration both causal and information dependencies in goal achievement. This type of planning is interesting, for instance, in smart home environments which contain various sensors and robots to provide services to the inhabitants. Requests for information, for instance from an activity recognition system, should cause the smart home to configure itself in such a way that all requested information will be provided when it is needed. This paper addresses temporal configuration planning in which information availability and goals are linked to temporal intervals which are subject to constrains.

Our solutions are based on constraint-based planning which uses different types of constraints to model different types of knowledge. We propose and compare two approaches to configuration planning. The first one models information via conditions and effects of planning operators and essentially reduces configuration planning to constraint-based temporal planning. The second approach solves information dependencies separately from task planning and optimizes the cost of reaching individual information goals. We compare these approaches in terms of the time it takes to solve problems and the quality of the solutions they provide.

Introduction

Smart home environments contain a large variety of sensors that can produce different information depending on how they are used and how their data is processed. To infer, for instance, if a person is sitting on a chair we could either use a pressure sensor installed in the chair or point a camera at that chair and analyze the camera's feed. Configuration planning is a form of task planning that explicitly considers information availability either as a goal or as a condition to use certain operators. A robot, for instance, may have to rely on a camera feed to navigate an apartment. In the context of the E-care@home project (ecareathome.se) we consider context inference that may require information to be available during specific temporal intervals and we employ configuration planning to ensure that these *information goals* are met.

Configuration planning is the combination of task-planning with information goals and dependencies. On one hand task-planning operators may require information to be executed (e.g., robot movement requires robot's location). In other scenarios information may be the goal. Consider a smart home environment equipped with a multitude of sensors that gather information about the inhabitant. Here an information goal could be ”Is the inhabitant eating at lunch time?”. There might be many alternative ways to get to this information. A robot could be used to determine the human activity with a camera during the requested time. Alternatively, information from contact sensors attached to the door of the fridge and pressure sensors attached to the seats in the kitchen could be used to come to the same conclusion. In addition to the dependencies themselves we may want to consider costs. Having a robot's camera detect a human activity for one or two hours is more expensive than using the alternative described above. As task planning can depend on available information, we may also have the opposite dependency, e.g., providing information with a specific sensor may require task planning. In the above example, the robot needs to be at a specific location in order to provide the requested information.

Constraint-based planning allows to model different types of knowledge via different types of constraints. The constraint-based planning problem is solved by integrating solvers for each of these constraint types. The presented approach already integrates temporal constraints, resources, costs, constraints for interaction with humans, and static knowledge via Prolog. This view of planning is interesting for configuration planning since it allows us to investigate different ways to capture information dependencies. The main contributions of this paper are two constraint-based solutions to temporal configuration planning. These are, to the best of our knowledge, the first approaches that consider the timing of information and temporal constraints on information goals. The first approach relies on task planning to model information goals and dependencies. Its main appeal is that it can rely on existing forward planning heuristics and produce solutions very quickly. However, it does not consider solution quality. The second approach utilizes problem decomposition to satisfy information dependencies as a sub-problem. This sub-problem is formulated as a Constraint Satisfaction Problem (CSP) with the constraint processing language MiniZinc (http://www.minizinc.org/). It then uses task planning to satisfy any open goals that result from the solution to the information sub-problem (e.g., sensors may have to be targeted at objects or be at specific locations). The main appeal of this approach is that it makes use of the optimization capabilities of MiniZinc to provides higher quality solutions. However, it takes significantly longer to produce solutions as we will see in the experimental section.

Related Research

There are many approaches where planning problems are encoded into SAT problems or CSPs (Kautz, Selman, and Hoffmann 2006; Kambhampati 2000). Constraint-based planning as described by (Köckemann, Pecora, and Karlsson 2014; 2015; Köckemann 2016) is instead an approach to planning that models different aspects of a domain with different types of constraints. This approach was used to provide solutions for several challenges of human-aware planning. In this paper we use the same framework but address a different problem. Other constraint-based planning approaches in a similar direction include EUROPA (Frank and Jónsson 2003) and SAM (Pecora and Cirillo 2009).

Configuration planning is a relatively recent topic that does not have a large body of related work. (Lundh 2009) describes an approach based on an extension of hierarchical task-network planning (Nau et al. 1999). Operators are extended to functionalities that describe information dependencies. The configuration planning problem is solved as a composition of the HTN planning problem and the problem of finding a configuration that satisfies all information dependencies. (Di Rocco, Pecora, and Saffiotti 2013; Di Rocco et al. 2013) go one step further and integrates information dependencies with causal, temporal and resource reasoning. Information is still treated as static in the sense that it does not consider information being available for only a limited temporal interval. The configuration planning problem with multiple, partially-ordered preferences was considered by (Silva-Lopez et al. 2015). There are also some similarities between configuration planning and forms of task allocation and coalition formation where sensing, communication and information conversion are considered (Zhang and Parker 2012).

Configuration Planning

Configuration planning is a combination of task planning which attempts to reach a set of goals by applying operators (or actions) with information dependencies and information goals. This can be viewed as an extension of the task planning problem, where operators may require information to be applicable (*information dependency*). A robot, for instance, may require a camera feed of a ceiling camera in order to move safely through the environment. On the other side gathering information might be the main goal (i.e., an *information goal*) that may require task planning to be achieved. Information dependencies can be represented by information links of the form

$$I \xrightarrow[TR]{\mathrm{c}} O \tag{1}$$

that require a set of information inputs I in order to provide some output information O at a cost c per time unit that the link is used. The set TR contains any non-information sub-goals (*task-requirements*) that need to be achieved to use the link. Basic information links allow to produce information directly from sensors ($I = \emptyset$). Using sensors may require the planner to fulfill task requirement subgoals. For instance, in order to determine if a human is executing a specific activity (e.g., eating) we may need to move a robot to the human so that the robots camera can detect the activity. We refer to these sub-goals as of information links. Note that we only consider whether or not information is available and that the information content has no impact on the planning problem. For instance, we may require access to a map of the environment (information goal) to navigate a robot, but the plan is not contingent on what information is contained in the map. That means that we are not required to deal with uncertainty about what will actually be observed during execution. Configuration planning deals with information availability but not with the information content.

Informally, a solution to a configuration planning problem is a selection of information links with associated temporal intervals that provide all information goals at the correct times and have no unsatisfied inputs. In addition, all task requirements need to be fulfilled.

Figure 1 shows a graph representation of a set of information links. All s_x in the figure represent sensor information provided by some sensor s_x. All i_x represent information that is computed from sensors or form other information. The graph is directed and acyclic and we consider sensors to be the original source of all information. Sensor $s2$, for instance, can provide information $i4$ at a cost of 43 per time unit. Sensor $s1$ can be used at a cost of 71 to provide information $i10$. To use sensor $s1$ for this purpose it needs to be set to configuration $cfg1$. In the same way sensor $s5$ needs to target object $o2$ in order to provide information $i2$ and $s3$ needs to be at location $l1$ to provide information $i2$. Information $i2$ and $i4$ can be combined to provide information $i5$ at a cost of 16. This example was randomly generated with the same script that was used for the evaluation later on.

Explicitly modeling information dependencies and goals as part of the environment has the advantage that the source of information could be changed dynamically in case sensors are not available due to hardware failure.

Previous work on configuration planning did not consider temporal aspects of information. Here we use constraint-based planning with temporal reasoning to model information availability during flexible temporal intervals. The following section gives a short introduction to constraint-based planning.

Constraint-based Planning

A promising framework for performing configuration planning is constraint-based planning. The approach to constraint-based planning that we use in this paper models different types of knowledge via different types of constraints. This allows us to propose and compare two very different methods for configuration planning within the same framework. Solvers for constraint-based planning problems can be composed of solvers for individual constraint types.

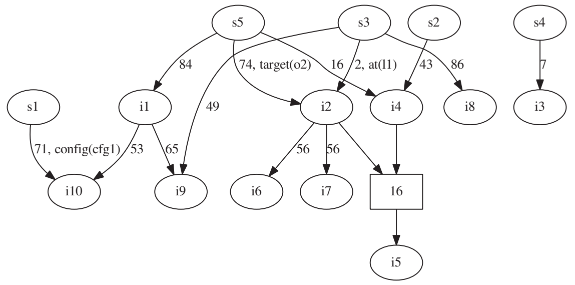

Figure 1: Example information dependencies. Edge labels are costs followed by requirements (if any exist). Links with multiple inputs are box shaped vertices where ingoing edges are required inputs, the single outgoing edge is the derived information, and vertex labels are link costs. We use the prefix s for raw sensor data, i for information, o for object, and l for location.

Each of these types poses a sub-problem and solving all sub-problems leads to a solution to the overall problem. A sub-problem for a constraint type can have three different outcomes. It can be *satisfied*, *unsatisfiable*, or require to *resolve a flaw*. In the latter case there might be a choice of resolvers that need to be explored systematically. For each constraint type we assume a solver that decides if a sub-problem can be resolved and how. This search over resolvers of flaws for all constraint types defines the constraint-based planning problem as a search problem. This search is realized by Algorithm 1 described later.

This view on planning is interesting for configuration planning since it allows us to investigate different ways to capture information dependencies. In this paper we will compare two such ways: modeling information dependencies into operator preconditions and effects directly, and modeling them with a distinct type of constraint that is solved separately by a constraint solver.

Compared to planning approaches using the Planning Domain Definition Language (PDDL) constraint-based planning has the advantage of using explicit temporal intervals which allow to easily state constraints on the intervals during which goals have to be achieved. This is especially relevant for information goals that need to be achieved during a fixed interval. While it might be possible to model our goal-based approach with PDDL, talking about time in temporal PDDL is surprisingly difficult and cumbersome and a comparison between our two approaches would become difficult.

A *Constraint Database (CDB)* Φ is a set of constraints of different types. Constraint types considered in this paper are statements, arithmetic (math) and cost constraints, temporal constraints, open goals, and interaction constraints. We will go over each type in turn. Later, we also introduce a new type of constraint for configuration planning that will be utilized in our second approach.

The most basic type is *statement*. Statements relate state-variable assignments to temporal intervals. They have the form $(\mathcal{I}\ x\ v)$ where $\mathcal{I}$ is a temporal interval, x is the state-variable and v the value. We use statements to model both the context for the task planner (initial state, preconditions, and effects), as well as information that is available during a temporal interval. Statements require scheduling whenever two statements with the same variable can have different values and their intervals have a non-empty intersection.

Example 1. *The first two statements below represent the location of a robot and an object. The third statement is used to represent which object the robot is facing. The last two statements actually represent information that is available during their intervals.*

```
(:statement (I1 (at robot) location)
  (I2 (at object) location)
  (I3 (facing robot) object)
  (I4 (camera-feed-raw robot object))
  (I5 (camera-feed-processed object)) )
```

A temporal context for statements is established by *temporal constraints* that can impose, e.g., flexible durations, release times or precedence constraints. We express temporal constraints via quantified Allen's interval relations (Allen 1984; Meiri 1996) between statements in CDBs. For convenience, we add disjunctions of conceptually neighboring constraints such as *during-or-equals* (Freksa 1992). We also add the unary temporal constraints *release*, *deadline*, *duration* and *at*.

Example 2. *The following temporal constraints applied to the statements from Example 1 state that the robot is facing the object while they are both at the same location. The raw camera feed is provided during the interval where the robot is facing the object. The two bounds state the difference between start and end times of the two intervals. Finally, the processed camera feed is provided with a temporal delay of 10 time units.*

```
(:temporal (during I3 I1 [1 inf] [1 inf])
  (during I3 I2 [1 inf] [1 inf])
  (during I4 I3 [1 inf] [1 inf])
  (overlaps I5 I4 [10 inf] [1 inf]) )
```

Goal constraints use statements to express open goals for the task planner to reach. Temporal constraints on the intervals of goal statements allow to define during what interval the goal has to be reached. Goal constraints require operators that define which statements can be reached from a CDB. An operator o is a tuple $(name, \mathcal{P}, \mathcal{E}, \mathcal{C})$ that consists of a name, a set of precondition statements $\mathcal{P}$ that must be in a CDB for the operator to be applicable, a set of effect statements $\mathcal{E}$ that are added when the operator is applied, and a set of constraints $\mathcal{C}$ that must be satisfied when the operator is applied. Usually, $\mathcal{C}$ contains temporal constraints that relate the temporal intervals of preconditions and effects to the operator itself. If an operator's effect adds a statement that can be matched to a goal that goal can be satisfied by connecting them with a temporal equals constraint (forcing effect and goal intervals to have the same solution space). In the next section we will see how operators can be used to establish information links.

Example 3. *The following operator requires a raw camera feed and produces a processed one. Information is represented by precondition and effect statements. The temporal overlaps constraint assures that the required information is available before processing (interval $?P$ has to start at least 10 time units before interval $?E$). The operators interval (represented by $?THIS$) is equal to the effect interval.*

```
(:operator (process-camera-feed)
  (:preconditions (?P (camera-feed-raw ?R ?O)) )
  (:effects (?E (camera-feed-processed ?O)) )
  (:constraints (:temporal (equals ?THIS ?E)
     (overlaps ?P ?E [10 inf] [1 inf]) ) ) )
```

The following goal can be achieved by the example operator if the raw camera feed is available during a temporal interval that can satisfy temporal constraints of both goal and operator. The temporal equals constraint is used to link effects with goals that they achieve (as shown below).

```
(:goal (G (camera-feed-processed object)) )
(:temporal (at G [0 50] [80 inf])
  (equals G I5) ) ;; G is satisfied by I5
```

Prolog constraints are used to express constraints on static facts. A Prolog program is provided together with the domain model and Prolog constraints are predicates used in that program. Here we use Prolog constraints in operators to express under which circumstances specific information can be derived in one of our solutions (i.e., we specify information dependencies in Prolog and assert then via Prolog constraints). We will see an example of this later.

Cost constraints are used to add up costs. In this paper we use them to model the cost of making information available. *Math constraints* can be used to perform calculations. Their only use in this paper is to calculate costs by using the duration of information intervals.

An *Interaction constraint* (IC) ic is composed of a condition CDB $Condition(ic)$ and a sequence of resolver CDBs $Resolvers(ic)$. If for any CDB Φ we have $ic \in \Phi$ and $\Phi \cup Condition$ can be satisfied, at least one $r \in Resolvers(ic)$ must be satisfiable. Previously, ICs were used to constrain the interaction with humans (Köckemann, Pecora, and Karlsson 2014). Here we use them to add goals when certain information links are used (only required by our second approach). An example will be discussed later. For a more detailed coverage of constraint-based planning see (Köckemann 2016).

To solve the constraint-based planning problem we use flaw resolution search. Flaw resolution is a very convenient way to integrate various types of knowledge. Two statements with the same variable but different values whose intervals intersect create a flaw that needs to be resolved by adding a temporal constraint that separates them in time. Open goals can be seen as flaws that need to be resolved via task planning.

Algorithm 1 shows the general approach we use to solve the constraint-based planning problem. It first uses optional preprocessing procedures for each constraint type (lines 3 and 4). Then it starts the recursive procedure RESOLVE-ALL that will test all constraint types in a given order (line 7) for flaws and inconsistencies with TESTANDRESOLVE (line 8). Note that detection and resolution of flaws is entirly done in TESTANDRESOLVE. If TESTANDRESOLVE returns *Failure* RESOLVE-ALL fails (line 10). If it returns Φ then Φ is accepted as a solution by type t. If flaws need to be resolved for type t RESOLVE-ALL returns a list of resolved CDBs that are tested recursively (lines 12 through 15). If any of these recursive calls returns a CDB then it is a solution (line 15). If none of them works the flaw cannot be resolved and *Failure* is returned (line 16). If no type t requires a change then Φ is returned as a solution (line 17).

Algorithm 1 Constraint-based planning

```
Require: $\Phi$ - a CDB, $\mathcal{O}$ - set of operators, $\mathcal{B}$ - Prolog knowledge
         base, $\Theta$ - an ordering of solvers
 1: function CB-PLAN($\Phi, \mathcal{O}, \Theta$)
 2:     global $\mathcal{O}, \Theta, \mathcal{B}$            ▷ Available to all sub-procedures
 3:     for $t \in \Theta$ do
 4:         PREPROCESS-$t$($\Phi$)             ▷ Optional preprocessing
 5:     return RESOLVE-ALL($\Phi$)
 6: function RESOLVE-ALL($\Phi$)
 7:     for $t \in \Theta$ do
 8:         $R \leftarrow$ TESTANDRESOLVE-$t$($\Phi[, \mathcal{O}][, \mathcal{B}]$)
 9:         if $R = Failure$ then
10:             return $Failure$                    ▷ $\Phi$ cannot be fixed
11:         if $R \neq \{\Phi\}$ then
12:             for $\Phi' \in R$ do
13:                 $\Phi'' \leftarrow$RESOLVE-ALL($\Phi'$)
14:                 if $\Phi'' \neq Failure$ then
15:                     return $\Phi''$   ▷ Solution from recursive call
16:             return $Failure$               ▷ No working resolver
17:     return $\Phi$                       ▷ No flaws: Solution found
```

Temporal reasoning is carried out in polynomial time by solving a *Simple Temporal Problem (STP)* with a variation of the all-pairs-shortest path algorithm (Floyd 1962). Scheduling of statements is handled by the approach described by (Cesta, Oddi, and Smith 2002). Prolog constraints are translated into queries that are solved with the Prolog solver *YAP*. Costs and math constraints do not require flaw-resolution and are reduced to simple evaluations. Interaction constraints are solved by the approach described by (Köckemann, Pecora, and Karlsson 2014). We use two approaches to resolve open goals. One is based in plan-space planning and implements the h_{add} heuristic with action reuse described by (Younes and Simmons 2003). Plan-space planning is straight-forward to use on top of our operator definition. (Note that threats are taken care of by temporal reasoning and scheduling.) The second one is based on a translation of CDB and operators into a variation of a forward state-space planning problem that is solved by alternating between the causal graph and fast forward heuristics (Helmert 2006; Hoffmann and Nebel 2001) together with the lookahead suggested by (Vidal 2011).The implementation is available open source at spiderplan.org.

Goal-based Approach

Our first approach models all information dependencies of the configuration planning problem as preconditions and effects of operators. This reduces temporal configuration planning to constraint-based temporal task planning. The operator *process-camera-feed*, for instance, requires the raw feed as a precondition and provides the processed feed as an effect. In general, links (Eqn. 1) are modeled with operators of the form $o_{link} = (I \cup TR, O, \mathcal{C} \cup \{(\text{:cost (add c)})\})$. Here, $\mathcal{C}$ also contains temporal constraints that assure that the output O is provided only while inputs I and task-requirements TR are satisfied.

The concrete operators used in this approach can be divided into three sets. *Sensor operators* deal with generating information from sensors (e.g., camera generates raw feed), *processing operators* deal with processing available information to generate new one (e.g., raw feed to processed feed), and operators that are used to fulfill task requirements of information links (e.g., by moving them or by targeting objects).

We consider four *sensor operators*: one without any preconditions (e.g., a fixed temperature sensor), one requiring the sensor to be at a specific location (e.g., a movable humidity sensor), one that it is targeting a specific object (e.g., a camera) and one that requires a specific configuration (e.g., sampling frequency). Targeting requires the sensor and the object to be at the same location. Other combinations of these task-requirements are also possible.

In addition, we consider six *processing operators*, each requiring between one and six information inputs to compute a single output information. Finally, we have a set of operators that address the task-requirements of sensor links. One operator allows to move sensors between different locations (if they are movable), one operator allows sensors to pick up and put down objects (these sensor may be considered as robots), two operators allow a sensor to start and stop targeting objects at their current location, and finally one operator allows to change the configuration of a sensor. This last set of operators is used as well by our second approach.

Unlike in the example operator *process-camera-feed* the sensor and processing operators we use in this approach rely on Prolog background knowledge to determine information dependencies and conditions. This background knowledge is central to the approach as it specifies the information links that are available. It allows us to specify operator templates (as in the following example).

Example 4. *Consider the following operator.*

```
(:operator
  (sense ?S - sensor ?I - info ?C- cost)
  (:effects (?E1 (available ?I))
    (?E2 (sensor-state ?S) on) )
  (:constraints (:temporal
      (equals ?THIS ?E1)
      (during ?THIS ?E2 [1 inf] [1 inf]) )
    (:prolog kb (linkSensor ?S none ?I ?C))
    (:math (eval-int (cost-term ?E1) (mult (
        sub (EET ?E1) (LST ?E1)) ?C)))
    (:cost (add link-cost (cost-term ?E1))) ) )
```

It does not have any task requirements ($TR = \emptyset$) so there are no preconditions. It turns on sensor ?S *and makes information* ?I *available. It calculates the cost by taking the minimum duration of* ?E1 *and multiplying it with the cost per time unit* ?C. *To determine which sensors can be used to make certain information available it uses a Prolog constraint.*

Every line in the Prolog knowledge base kb *that matches this constraint determines an information link without task requirements. Consider the following Prolog code that represents three of the links from Figure 1.*

```
linkSensor(s1,config(cfg1),i10,71).
linkSensor(s5,none,i1,84).
link2(i5,i2,i4,16).
```

Sensor s1 *can be used to produce information* i10 *with cost 71 per time unit when it has configuration* cfg1. *Sensor* s5 *can be used to produce information* i1 *with cost 84 per time unit without any special condition. This allows to apply the basic sensor operator to use this link. Information* i5 *can be produced using* i2 *and* i4 *for a cost of 16 per time unit.*

This approach captures all information dependencies by modeling them as part of a task planning problem. Our second approach attempts to model information goals as a separate problem that can be solved with constraint processing.

CSP-based Approach

This approach models information goals and dependencies as a separate type of knowledge. We introduce a new type of constraint that allows to model information dependencies via links with attached costs and to specify information goals. This kind of solution is enabled by Algorithm 1, since we just need to add a new type t and provide the method TESTANDRESOLVE-t. This solver is integrated with the constraint-based planner shown in Algorithm 1.

The new type *configuration-planning* uses the following elements. An information dependency is modeled with a link

```
(link O I c)
```

that models Eqn. 1 where O is the output, I is the required input, and c is the cost per time-unit of using the link.

Example 5. *The knowledge base used in Example 4 would be represented as follows.*

```
(link i10 {s1} 71)
(link i1 {s5} 84)
(link i5 {i2 i4} 16)
```

Note that special conditions are not covered in these links. We will see how they are added at the end of this section.

Information goals are modeled as

```
(goal interval target)
```

and include a temporal *interval* that can be used to determine when the *target* information has to be available. While we consider information goals as the main purpose for planning in this paper, this approach can also be used to model operators that require information to be used. In this case one simply needs to add an information goal to the set of constraints of the operator (recall that open goals are seen as constraints in our approach).

The configuration planning solver converts links into input data for a CSP formulated in the language MiniZinc. It solves one information goal at a time while minimizing the costs of the involved links. The CSP for a single information goal attempts to find the cheapest selection of links to provide the information goal. Each selected link must also have all its requirements satisfied. The only information sources without information sub-goals are sensors. For each information link that is chosen, a statement is added together with temporal constraints to assure that all input information is available when it is needed. The CSP does not consider requirements of sensors such as targeting. Instead it uses *Interaction Constraints* (ICs) to add goals in case links with attached conditions are used. These goals are then taken care of by a task planner that uses plan-space planning. The reason we use an open goal solver based on plan space planning is that we attempt to resolve interaction constraints and open goals after each satisfied information goal. The planner used in the goal-based approach is efficient because it is based on state-space planning but this also makes it harder to modify existing plans.

Example 6. *Consider the first link of Example 5. As mentioned before, compared to Example 4, we are missing the required configuration of this link. We address this problem in the following way. If a link is used by the configuration planning solver, a statement will be added to the CDB.*

```
(:statement (I (link i10 {s1} 71)))
```

The condition of the following IC checks if such a statement exists in a CDB and if it does exist its resolver requires to add a goal for the sensor to have the required configuration at the right time.

```
(:ic (link-requires-targeting ?I s1 i10)
 (:condition (:statement (?I (link i10 {s1} 71))))
 (:resolver (:goal (?G (config s1) cfg1))
  (:temporal (during ?I ?G [1 inf] [1 inf]))))
```

The benefit of this approach is that it isolates the sub-problem of satisfying information dependencies and allows to reduce the cost of reaching information goals. Note that the resulting solution is not necessarily globally optimal since one information goal is satisfied at a time and a globally optimal solution would need to solve all information goals at once and possibly have to consider temporal information since costs are per time-unit of the intervals during which information links are used. The resulting task planning problem contains only goals that actually require task planning. The drawback is that links with competing conditions may be chosen. In such cases it could take Algorithm 1 a long time to realize that no plan can be found for a selected set of links. Since the search is systematic, however, a solution will be found eventually.

Experiments

We compare the two presented approaches wrt. solving time and solution quality. In order to do so, we created three random domains and for each of these domains we created 200 problems for both presented approaches. Parameters for domain generation are the number of sensors (20, 30, 40), information types (60, 90, 120), information links (100, 150, 200), objects (20), locations (10), sensor configurations (5), information cost (uniformly random between 0 and 100), goal start time (uniformly random between 100 and 100000), goal minimum duration (uniformly random between 50 and 100), required targeting conditions (2,3,4), required location conditions (2,3,4), required configuration conditions (2,3,4). Since costs of information links are chosen randomly they are not influenced by the fact that they may have special conditions. In a real-world domain we would expect that using sensors with such conditions (likely robots) would usually come at a higher cost than using a stationary sensor. All domain, problem, and planner definitions used in this paper can be found online (spiderplan.org).

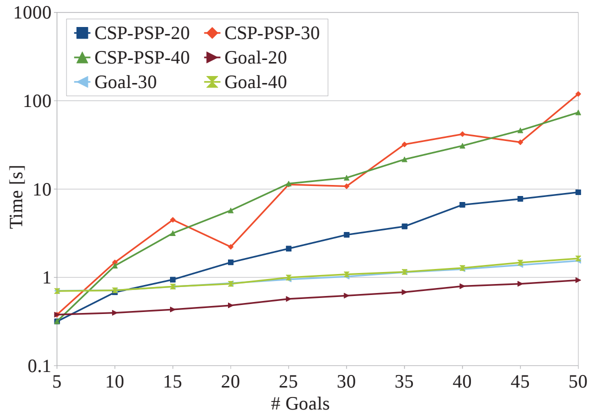

Figure 2: Average time to solve 20 problems for a varying number of information goals.

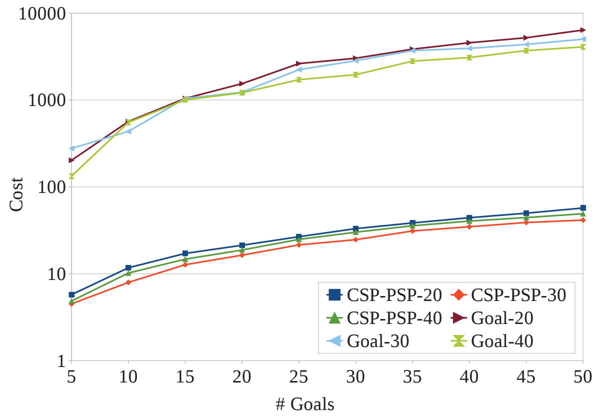

Figure 3: Average cost of the solutions to 20 problems for a varying number of information goals.

For each domain we generated 20 problems for a varying number of information goals ($5, 10, \ldots, 50$) resulting in a total of 200 problems per domain. For each problem we generated a goal-based version and a CSP-based version that contain exactly the same goals. We refer to our two approaches as *Goal* and *CSP*. Results for approach and domain combinations are written *Goal-20*, *Goal-30*, *Goal-40*, *CSP-20*, *CSP-30*, and *CSP-40* (using the number of sensors in the domain).

Figure 2 shows the average time to solve 20 random problems for different numbers of information goals. We can see that the solving time is clearly superior for the goal based approach. This is not surprising since the goal-based approach uses very efficient planning heuristics while the CSP-based approach relies on plan-space planning which requires resolving threats. Figure 3 shows the average costs of the solutions. The problems generated for each domain are the same for both approaches, so costs for *Goal-x* and *Sensor-x* are comparable. Costs are much lower for the CSP-based approach. This can be explained by the fact that the goal based approach does not consider optimization at all. If a piece of expensive information is required once very early and once at a later point in time, both goals can be satisfied by one interval. If in a real-world example we need a camera to see if a person is sleeping in the morning and in the evening, the goal-based approach would use the camera the entire day to satisfy both goals. While not globally optimal, the CSP-based approach optimizes individual information goals and thus does not suffer from this issue. Solving one goal at a time would not have the same effect for the goal-based approach, since it would simply extend an existing information interval to fit a goal if possible.

Conclusions & Future Work

We employed constraint-based planning to introduce and compare two approaches to temporal configuration planning. Our results confirm our intuition that the goal-based approach finds solutions very efficiently, while the CSP-based approach finds higher quality solutions but may require more time to find them.

For future work it will be interesting to investigate ways to combine the advantages of both presented approaches. In addition we will test the approach in a real-world setting for context-recognition in one of the E-care@home smart-home environments where it will receive goals from a context-recognition system and extract information links and costs from a sensor ontology. Another important aspect of configuration planning that we would like to address in the future is re-planning and plan repair in case sensors become unavailable for any reason.

Acknowledgements

This work was supported by the Swedish Knowledge Foundation (KKS) project "E-care@home".

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23:123–154.

Cesta, A.; Oddi, A.; and Smith, S. 2002. A Constraint-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics* 8:109–136.

Di Rocco, M.; Pecora, F.; Sivakumar, P.; and Saffiotti, A. 2013. Configuration planning with multiple dynamic goals. In *AAAI Spring Symposium*.

Di Rocco, M.; Pecora, F.; and Saffiotti, A. 2013. When robots are late: Configuration planning for multiple robots with dynamic goals. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 9515–5922. IEEE.

Floyd, R. W. 1962. Algorithm 97: Shortest Path. *Communications of the ACM* 5:345–345.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8:339–364.

Freksa, C. 1992. Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence* 54:199–227.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of AI Research* 14:2001.

Kambhampati, S. 2000. Planning graph as a (dynamic) CSP: exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of AI Research* 12:1–34.

Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In *In Proceedings of the 5th International Planning Competition (IPC)*.

Köckemann, U.; Pecora, F.; and Karlsson, L. 2014. Grandpa hates robots — interaction constraints for planning in inhabited environments. In *Proceedings of the 28th Conference on Artifical Intelligence (AAAI 2014)*.

Köckemann, U.; Pecora, F.; and Karlsson, L. 2015. Inferring context and goals for online human-aware planning. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*.

Köckemann, U. 2016. *Constraint-based Methods for Human-aware Planning*. Ph.D. Dissertation, Örebro university.

Lundh, R. 2009. *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. Ph.D. Dissertation, Örebro University, School of Science and Technology.

Meiri, I. 1996. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Artificial Intelligence*, 260–267.

Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 968–975.

Pecora, F., and Cirillo, M. 2009. A constraint-based approach for plan management in intelligent environments. In *Proceedings of the Scheduling and Planning Applications Workshop (SPARK) at ICAPS 2009*.

Silva-Lopez, L. S. d. C.; Broxvall, M.; Loutfi, A.; and Karlsson, L. 2015. Towards configuration planning with partially ordered preferences: Representation and results. *KI* 29(2):173–183.

Vidal, V. 2011. YAHSP2: Keep It Simple, Stupid. In *Proc. of the Int'l Planning Competition (IPC-7)*.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research* 20:405–430.

Zhang, Y., and Parker, L. E. 2012. Task allocation with executable coalitions in multirobot tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 3307–3314. IEEE.