Incremental Heuristic Search for Planning with Temporally Extended Goals and Uncontrollable Events

Adi Botea*
NICTA† and
the Australian National University
Canberra, ACT

André A. Ciré
Institute of Computing
University of Campinas
Brazil

Abstract

Planning with temporally extended goals and uncontrollable events has recently been introduced as a formal model for system reconfiguration problems. An important application is to automatically reconfigure a real-life system in such a way that its subsequent internal evolution is consistent with a temporal goal formula.

In this paper we introduce an incremental search algorithm and a search-guidance heuristic, two generic planning enhancements. An initial problem is decomposed into a series of subproblems, providing two main ways of speeding up a search. Firstly, a subproblem focuses on a part of the initial goal. Secondly, a notion of action relevance allows to explore with higher priority actions that are heuristically considered to be more relevant to the subproblem at hand.

Even though our techniques are more generally applicable, we restrict our attention to planning with temporally extended goals and uncontrollable events. Our ideas are implemented on top of a successful previous system that performs online learning to better guide planning and to safely avoid potentially expensive searches. In experiments, the system speed performance is further improved by a convincing margin.

1 Introduction

Many real-life systems, such as a power grid, a network of water pipes, or the machinery inside a factory, need to be configured such that their subsequent functioning, called the nominal behaviour, is the desired one. The problem can be formalized as an extended planning model, which handles both temporal logic goals and events, transitions that are not under the control of a planning agent. Configuration steps are represented as actions under the control of the planning agent. As soon as a configuration process completes, the system starts to function on its own (nominal behaviour). Transitions during the nominal behaviour are modelled as uncontrollable events. The nominal behaviour is a non-deterministic process. If several events are applicable in a state, only one of them, selected at random, will occur. The planning task is to configure a system in such a way that every possible evolution (event sequence) in the nominal behaviour is consistent with a goal formula expressed in temporal logic.

Despite its importance, the problem of reconfiguring a system to achieve a desired subsequent nominal behaviour has received little attention in the AI literature. In addressing this need, this paper presents the following main contributions.

We introduce an incremental planning algorithm based on goal decomposition, and a search-guiding heuristic that estimates the relevance of an action with respect to a goal. The relevance heuristic is fast to compute at runtime, as all the needed information can be obtained in a pre-processing step. Even though our heuristic might seem somewhat simpler than current heuristics used in classical planning, it has the advantage that it works for temporally extended goals. Adapting other heuristics from reachability goals to temporally extended goals that model the nominal behaviour is by no means a trivial task.

The incremental algorithm and the relevance heuristic are not limited to planning with temporal goals and uncontrollable events. However, in this work we restrict our attention to this planning model. Further application ideas are briefly discussed in the future work section. Our enhancements are implemented on top of an existing solver for planning with temporal goals and uncontrollable events. The basic solver is already fast, as it contains an online learning step that helps guide the planning and avoid potentially expensive (model-checking) searches. See a brief overview in Section 3. In experiments, the enhancements presented in this paper further improve the solver speed by a convincing margin.

2 Related Work

The area of configuration relates to composing pre-defined parameterisable components in order to satisfy a set of constraints and some desired requirements [Mittal and Frayman, 1989; Sabin and Weigel, 1998]. AI approaches have been established as central technologies to the configuration paradigm, as they provide the appropriate formalism and efficient reasoning methods for coping with large configuration systems and complex component interactions. A distinctive

*We thank the annonymous reviewers for their feedback.

†NICTA is funded through the Australian government's *baking Australia's ability* initiative.

feature of our work is that we consider the future evolution of the system as part of the task objective.

Planners such as TLPLAN [Bacchus and Kabanza, 2000] use temporal logic to represent search control knowledge. More recently, temporal logic has been used to express hard and soft constraints (preferences) in a plan. Examples of systems that implement this paradigms are SGPLAN [Hsu *et al.*, 2006], MIPS [Edelkamp *et al.*, 2006] and HPLAN-P [Baier *et al.*, 2007]. Temporal logic has been integrated in reactive planning, where actions must respond to event occurrences (e.g., [Barbeau *et al.*, 1998]). Differently from these contributions, the temporal information in our configuration problem refers only to transitions in the future nominal behaviour, not to actions controllable by the planning agent. Moreover, there is no goal state to reach in reactive planning, while our model has a clear notion of a goal state.

The incremental search algorithm can be seen as a generalization of planning with *goal agendas* [Koehler, 1998]. Goal agendas require completeness conditions that often cannot be satisfied in practice. For this reason, one of the most popular goal agenda implementations, integrated in the FF planning system [Hoffmann and Nebel, 2001], gives up on completeness. In contrast, our approach is complete, as it allows backtracking from one subproblem to another in case a (local) solution was not found.

Our incremental planning approach fits into the broader area of *factored planning*. The idea in factored planning is to explore loose planning domain interactions in a divide-and-conquer fashion, aiming at improving the search efficiency. Recent approaches, such as [Amir and Engelhardt, 2003; Kelareva *et al.*, 2007], are based on a representation of the subdomains into a tree-like structure that captures the interactions among them. Brafman and Domshlak [2008] formalize the concept of coupling in a multi-agent planning system, and introduce a problem decomposition method based on a partitioning of the set of actions. In contrast, we focus on an iterative decomposition of the temporally extended goal.

Partitioning an initial pool of actions into relevant and irrelevant actions is a typical preprocessing step implemented in most modern planning systems. The goal is to reduce the search effort by ignoring actions that are either provably or heuristically ruled out as irrelevant to the problem at hand. Our relevance heuristic extends this idea to get a finer distinction between relevant actions. Actions are assigned relevance scores that are useful to heuristically guide a search.

3 Background

The formal planning problem we address has been introduced in previous work [Ciré and Botea, 2008]. We briefly reproduce the definition. A planning task is a structure $\langle S, s_0, \varphi, \gamma, A, E\rangle$ with S a finite state space, $s_0 \in S$ an initial state, and φ a temporal logic formula that describes the goal. The function $\gamma : S \times (A \cup E) \rightarrow S$ models deterministic transitions in the state space. The transitions are partitioned into a set of actions A (i.e., transitions under the control of the planner), and a set of uncontrollable events E that define the nominal behavior of a system. As in STRIPS planning, each transition a has a set of preconditions pre(a), a set of add effects add(a) and a set of delete effects del(a).

The state (sub)space $\mathcal{P}(s)$ generated from a given state s by applying only actions is called the *planning space* of s. Likewise, the (sub)space $\mathcal{M}(s)$ generated from s by applying only events is called the *model checking space* of s.

The planning task is to find a finite sequence of actions that can be applied to s_0 and that reaches a goal state. A state is a goal state if it *satisfies* φ. By definition, a state $q \in S$ satisfies a temporal logic formula ϕ iff every event sequence that originates in q (i.e., is applicable to q) satisfies ϕ.

We add our contributions on top of an existing solver [Ciré and Botea, 2008], which we'll call CB in the rest of this paper. A brief description and analysis of CB are necessary for a better understanding of the remaining sections. CB performs a global search in $\mathcal{P}(s_0)$, the planning space of the initial state s_0. To check whether a state $s \in \mathcal{P}(s_0)$ is a goal, a *model checking search* is performed in $\mathcal{M}(s)$. If all event sequences that originate in s are consistent with φ, then s is a goal state. Otherwise, an online learning step analyzes event sequences where φ does not hold. A condition c is extracted that holds in s and explains the failure of φ along such an event sequence. All states where c holds are guaranteed not be goal states. All conditions c are disjunctively combined into the global learned information I.

A main benefit of using I is the ability to eliminate unnecessary model checking searches. There is no need to run a model checking search in a state s where I holds, as s is guaranteed not to be a goal state. As a second benefit, I can guide the global search in the planning space. $\neg I$ can be seen as a weak form of a reachability goal, which indicates what states to avoid instead of indicating what states to reach. An additional complication raises from the fact that I evolves in time, and therefore different states could be evaluated using different reachability goals $\neg I$. For such reasons, $\neg I$ is not used not to compute heuristic estimates of the distance to a goal state. It is used only to partition the planning space into helpful states and rescue states [Hoffmann and Nebel, 2001; Vidal, 2004], a process that we call *helpful partitioning*. As a result, the savings achieved in the global search appear to be more limited than the savings in the model checking component. Our contributions, outlined in the following sections, address this limitation.

4 Incremental Planning

The search in the planning space is decomposed into a series of subproblems. Assume the global goal is expressed as a conjunction of subformulas $\varphi = \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_n$. A subproblem is created for each subgoal $\phi_k = \varphi_1 \wedge \cdots \wedge \varphi_k$. That is, the algorithm tries to achieve the first subgoal, then achieve the first two subgoals and so on. For each $k \in \{1 \ldots n\}$, the corresponding subproblem is a local search S_k that starts from a state that satisfies ϕ_{k-1} and seeks a state that satisfies ϕ_k. (Assume $\phi_0 =$ true.) A state that satisfies ϕ_k is called a local goal state for S_k. A local search S_k can be interrupted, when searching in other subproblems becomes necessary, and possibly resumed later. An interruption is performed to either advance to the next subproblem or to backtrack to the previous subproblem. The search performed in a subproblem be-

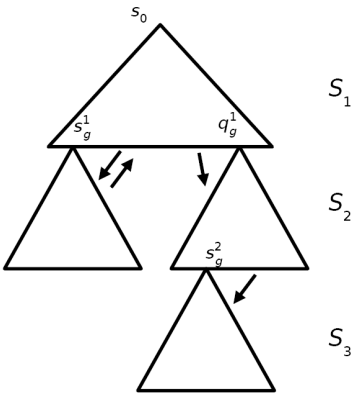

Figure 1: Example of incremental planning. Arrows indicate advancing and backtracking from one subproblem to another.

tween two consecutive interruptions is called a *search round*. All searches share a global closed list and a global transposition table for duplicate detection. Each subproblem S_k has its own open list O_k. An open list O_k is ordered using the helpful partitioning as a main criterion and the relevance heuristic value as a tie-breaker.

Figure 1 illustrates how incremental search works. A search round for S_1 is launched from s_0 to seek a state where φ_1 is satisfied. As soon as a local goal state s_g^1 is found, S_1 is interrupted. Its open list is not discarded, since this search might be resumed in the future, as shown later in this example. A search round for S_2 is launched from s_g^1 to seek a state where $\phi_2 = \varphi_1 \wedge \varphi_2$ is satisfied. In our example, assume that, during this search round, S_2 explores the entire subtree of s_g^1 without finding a local goal state. Note that this is equivalent to the O_2 open list becoming empty. As no local goal state was found in the most recent search round, the algorithm backtracks to S_1 and resumes it to seek another state that satisfies φ_1. As soon as a new local goal state q_g^1 is found, it is added to O_2 and S_2 is resumed. Since the previous search round in S_2 ended up with an empty open list, the new search round is rooted in q_g^1. The process continues in this fashion until either a search round in S_n finds a solution (global success) or S_1 ends up with an empty open list (no global solution). The steps of the incremental search algorithm are outlined more formally in Algorithm 1. See the explanatory comments and the previous example to follow the pseudocode more easily.

Algorithm 2 describes a search round in a subproblem. At line 2, it enumerates *candidate goal states*, states that could possibly satisfy the local temporal goal ϕ_k. Notice that a local search is allowed to temporarily destroy the previous goal ϕ_{k-1}, but in such a case it must re-achieve it in order to satisfy the stronger goal ϕ_k. Processing a candidate goal state (lines 5–11) is similar to how CB processes states in its global search. As outlined in Section 3, a model checking search is run to test whether the candidate goal state at hand is a (local) goal state indeed (line 7). The model checking search is safely skipped when possible (lines 5–6). If a model checking search returns a failure, a new online learning step is triggered

Algorithm 1 Incremental planning.

```
1:  I ← false {initialize learned info (global variable)}
2:  k ← 1 {index of the subproblem at hand}
3:  s_0 → O_1 {initialize open list of first subproblem}
4:  s_0^1 ← s_0 {root of search in first subproblem is s_0}
5:  while true do
6:      (s_g^k, π^k) ← SearchRoundInSubproblem(k) {return
        both final state and plan}
7:      if π^k = nil then
8:          k ← k − 1 {backtrack to previous subproblem}
9:          if k = 0 then
10:             return no global solution
11:     else
12:         if k = n then
13:             return π^1; π^2; ...; π^n {global success}
14:         else
15:             s_g^k → O_{k+1}
16:             s_0^{k+1} ← s_g^k {root of new search round is s_g^k}
17:             k ← k + 1 {move on to next subproblem}
```

(line 11).

To enumerate candidate goal states (line 2), the planning space $\mathcal{P}(s_0^k)$ is explored, where s_0^k, the current local root, is set as shown in Algorithm 1, lines 4 and 16. The exploration is guided with both helpful partitioning and a new relevance heuristic, described in detail in Section 5. As in CB, helpful states have higher priority, and a rescue state is explored only when no helpful states are currently contained in the open list. This is similar to using a heuristic that can take only two possible values (one value for helpful states and another, lower-priority value for rescue states). A major limitation of helpful partitioning is that it cannot distinguish further between states that belong to the same partition. For example, all helpful states look equally good to the helpful partitioning heuristic. For this reason, CB performs blind search as long as the set of helpful states in the open list is not empty. We address this by adding a secondary ordering criterion, given by the relevance heuristic. The relevance heuristic can take many possible values, and therefore states can potentially be ordered much more precisely.

Algorithm 2 SearchRoundInSubproblem.

```
1:  while true do
2:      (s_g^k, π^k) ← GetNextCandidateGoalState() {π^k is the
        action sequence from s_0^k to s_g^k}
3:      if no state s_g^k is found then
4:          return (nil, nil) {no solution}
5:      if s_g^k ⊨ I then
6:          continue {no need for a costly mod. check. round}
7:      ModelChecking(s_g^k) {check if s_g^k satisfies φ_k}
8:      if model checking succeeds then
9:          return (s_g^k, π^k)
10:     else
11:         I ← I ∨ ExtractInfo() {learning}
```

As the closed list, the transposition table, and the learned information I are global, the effort spent in a search round can be reduced using data acquired in previous search rounds. When expanding a state, successors that have been visited before, either in the current or in a different local search are skipped. This ensures that, despite running multiple searches in the global planning space, a state does not have to be expanded more than once.

A global I allows to eliminate more model checking searches. The global information I decomposes as $I = I_1 \vee I_2 \vee \ldots I_n$, where I_k is the information learned while searching in the k-th subproblem, using ϕ_k as a local goal. States s where I_k holds are guaranteed not to satisfy ϕ_k. If the learned information were used only locally, then I would be replaced with I_k at line 5 of Algorithm 2. As I is weaker than I_k, the test at line 5 will succeed more often when I is used, skipping more model checking searches. As argued below, this choice has no negative effect on the completeness.

The incremental algorithm is sound and complete. The soundness is obvious, as a global success is reported only when a state that satisfies $\phi_n = \varphi_1 \wedge \cdots \wedge \varphi_n = \varphi$ is found. To better emphasize the completeness, we point out that the incremental algorithm can be seen as a global search algorithm, except for two main differences: (1) there are multiple open lists; and (2) when processing a state, the goal test is performed for a goal formula ψ that can be weaker than the global goal φ (i.e., $\varphi = \psi \wedge \alpha$, with α an arbitrary formula). We discuss each of these differences in turn.

The concatenation of the local open lists can be seen as a global open list where pop and push operations can be performed at arbitrary, yet well-specified locations. For example, when searching in the k-th subproblem, only the portion corresponding to O_k allows insertion and extraction operations. Clearly, such a global open list affects the direction in which the planning space is explored, but has no impact on the algorithm completeness.

The goal test for each visited state s is performed at lines 5 and 7–8 in Algorithm 2. If s is ruled out as a local goal at line 5 (case 1), it means that $s \not\models I$, which is equivalent to $s \not\models I_1 \vee I_2 \vee \ldots I_n$ and, further, to $(\exists j) : s \not\models I_j$. The last expression implies that s does not satisfy ϕ_j. Since $\varphi = \phi_j \wedge \varphi_{j+1} \wedge \ldots \varphi_n$, we obtain that s does not satisfy φ. If s is ruled out as a local goal at lines 7–8 (case 2), we obtain that s does not satisfy ϕ_k and, following a similar argument as above, s does not satisfy φ. In either case, a state ruled out as a (local) goal is guaranteed not to satisfy the global goal φ. Therefore, we never miss out solutions and the completeness is not affected.

5 Relevance Heuristic

Besides considering only a part of the global goal, a subproblem keeps the search focused by using a heuristic measure of how relevant actions are to the subproblem at hand. Informally, this relevance heuristic gives priority to actions that guide the search towards either local goal states (*exploitation*) or non-goal states where learning new information can be successful (*exploration*). The latter scenario is useful to discover information that would speed up the rest of the solving process. The sooner such information is learned, the greater the potential savings are.

Given a subproblem, all actions are assigned numerical values as a measure of their relevance. As illustrated later in this section, smaller values correspond to more relevant actions. As no actions are ruled out as totally irrelevant, the algorithm completeness is not affected. An action can have a different relevance to two different subproblems. However, the relevance of an action to a given subproblem is fixed from one search round to another. Therefore, the relevance can be obtained in a pre-processing step.

When computing the relevance, the only subproblem-specific information given as an input parameter is a temporal formula ψ. See details about how ψ is selected for each subproblem later in this section. The relevance is computed with an iterative, Dijkstra-like procedure called RC. RC maintains a list of atoms L that is initialized to $\mathcal{A}(\psi)$, the set of atoms in ψ, and gets extended with action and event preconditions. Initially, each action (or event) a has its relevance value $r_\psi(a)$ set to ∞. At a given iteration i, zero or more transitions (actions and events) a have their relevance $r_\psi(a)$ set to i and have their preconditions added to L. For this to happen, a transition a must satisfy both following conditions: (1) $r_\psi(a) = \infty$ and (2) $\mathrm{add}(a) \cap L \neq \emptyset$ or $\mathrm{del}(a) \cap L_1 \neq \emptyset$. L_1 is the subset of L that contains the atoms in ψ and the preconditions of the events marked as relevant so far: $L_1 = \mathcal{A}(\psi) \cup \{p | (\exists e \in E) : r_\psi(e) < \infty \wedge p \in \mathrm{pre}(e)\}$. The process stops at an iteration i_M where a fixpoint is reached. All actions a with $r_\psi(a) = \infty$ get assigned a relevance value of $i_M + 1$.

By definition, we say that an action a impacts the formula ψ if $(\mathrm{add}(a) \cup \mathrm{del}(a)) \cap L_1 \neq \emptyset$. This means that a can either change the truth value of some atoms in ψ, or it can affect the preconditions of an event sequence that can change the truth value of some atoms in ψ.

In a local search round, relevant actions set the grounds to apply actions that are even more relevant, up to the point where a relevant action impacts the formula ψ. This strategy, that guides the search towards making changes that impact the formula ψ, is more desirable than searching in an area where the changes in the current state have no relevance to the goal at hand. One main advantage is that a state where ψ holds is more likely to be achieved. Furthermore, the changes in the current state produced by actions that impact ψ can trigger more activations of the learning procedure and hence end up in having more comprehensive information available.

For the subproblem S_k, the input parameter ψ given to RC is set to φ_k instead of the entire subgoal $\phi_k = \phi_{k-1} \wedge \varphi_k$. We have made this choice based of the following observations. Since ϕ_{k-1} is satisfied by searching in the previous subproblems, the current search round should focus on achieving φ_k without temporarily breaking ϕ_{k-1}, unless this is necessary. In addition, computing the relevance with respect to a smaller formula keeps a search more focused due to the following *monotonicity* property. Consider two temporal formulas ψ_1 and ψ_2 with $\mathcal{A}(\psi_1) \subseteq \mathcal{A}(\psi_2)$. For a given relevance value v, let $A_\psi(v)$ be the set of all actions whose relevance with respect to ψ is v. Then it can easily be shown that $\bigcup_{i \leq v} A_{\psi_1}(i) \subseteq \bigcup_{i \leq v} A_{\psi_2}(i)$. In other words, using a

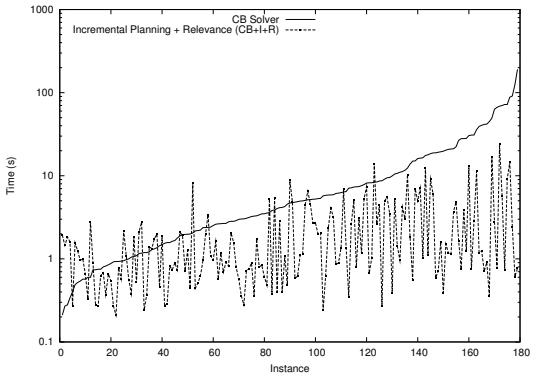

Figure 2: Time on a logarithmic scale.

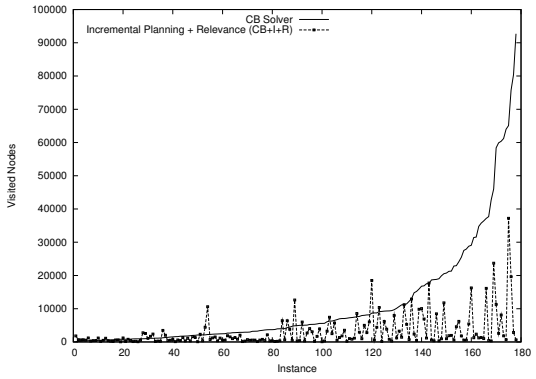

Figure 3: Visited nodes in planning.

smaller formula tends to build a smaller pool of actions that have a higher priority, focusing the search more effectively.

6 Experimental Results

We have implemented the incremental algorithm and the relevance heuristic in Java 1.6, on top of the CB solver [Ciré and Botea, 2008]. The domain [Ciré and Botea, 2008] used in experiments is available online at `http://abotea.rise.anu.edu.au/factory-benchmark/`. An instance has machines and repositories, as in a factory. Planning actions add and remove connections between machines and repositories, and clean machines after having used certain raw products. Uncontrollable events include the automatic transfer of raw material from repositories to machines, and the creation of final products by combining specific raw materials. The nominal behaviour (LTL goal) states that certain raw materials must never mix (otherwise the machine would break down) and that certain finite products must eventually be produced. There are 350 instances, with the number of actions and events ranging each from 50 to 150. The temporal formula size ranges from 5 to 15 conjunctive clauses. As we didn't study goal ordering heuristics yet, sub-goals are considered in the order they appear in the input file.

The experiments are peformed on an Intel 3.2 GHz machine, with 4 GB RAM. The time is set to 30 minutes for each instance. The planner stops as soon as the first solution is found or the problem is proven to have no solution. The experiments focus on a few directions. First, we compare the CB solver with the enhanced solver. Then, we analyze how each of our two main enhancements contribute to the performance. Finally, the effect on the solution quality is evaluated.

To save room, we will focus only on instances where a solution exists. For the remaining ones, there is no significant difference in performance among the planners. This is expected to happen, since the numbers of visited nodes in planning are necessarily the same for all solvers in such instances.

Figure 2 compares the original CB and the enhanced solver (CB+I+R). In very easy problems, both solvers are fast. Except for some easy problems, CB is almost never faster. As problems grow, the speed difference tends to increase, and CB+I+R rarely needs more than 10 seconds to solve a problem in this set. The largest improvement over CB exceeds

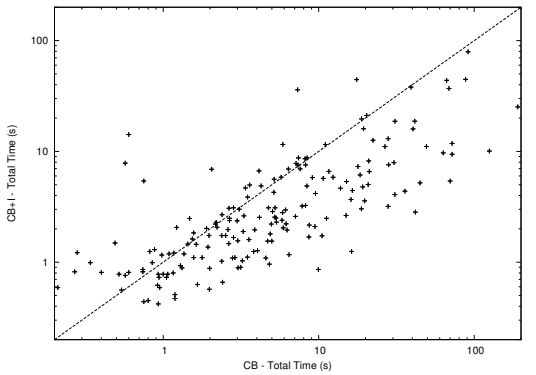

Figure 4: Impact of incremental planning algorithm.

two orders of magnitude. This gain in performance is a result of both visiting fewer states in the planning space, as shown in Figure 3, and performing fewer model checking rounds.

Next we analyze the speed-up gain contributed by each of the two enhancements individually. Figure 4 presents a time comparison between global search (CB) and incremental search (CB+I). Each data point corresponds to a problem instance. The x-coordinate is the time required by CB, whereas the y-coordinate corresponds to CB+I. If a data point is below the diagonal, then CB+I is faster in that instance. Most data points fall below the diagonal. The only cases when CB is faster correspond to relatively easy instances.

To illustrate the impact of the relevance heuristic, Figure 5 presents a comparison between global search (CB) and global search enhanced with the relevance heuristic (CB+R). Similarly to the behaviour of the incremental algorithm (Figure 4), the relevance heuristic significantly speeds up the solver in many cases. Most instances that are challenging for CB become quite easy with the heuristic in use.

Plan length is improved with a simple post-processing step. Even though plans are generated as a totally ordered sequence, they can be generalized to a partial ordering, with many possible total orderings. Our procedure attempts to rearrange a plan into a sequence that contains cycles (state repetitions). The action sequence between two identical states is

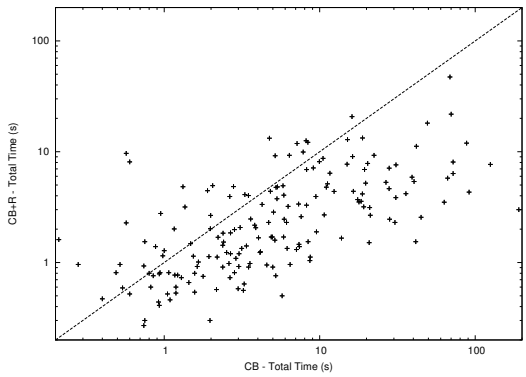

Figure 5: Impact of relevance heuristic.

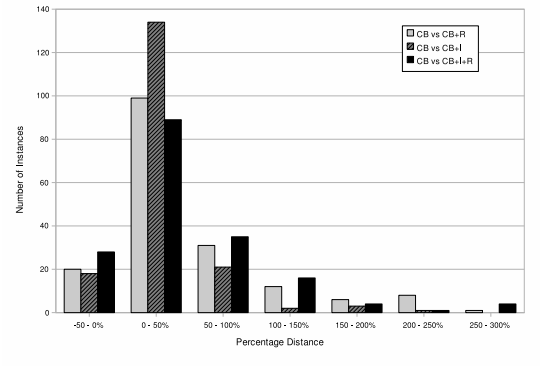

Figure 6: Impact on plan length.

removed. The process continues as long as possible.

Figure 6 summarizes the differences in plan length between CB and the three enhanced versions (CB+R, CB+I, CB+I+R). Given an enahced solver version Y, the percentage distance for an instance is computed as $(s_Y - s_{CB})/s_{CB} * 100$, where s_X is the plan length obtained by planner X after post-processing. Even though the impact on plan length varies in both directions, the enhanced planner versions tends to produce longer plans than CB. In two thirds of the cases, the percentage distance stays within the $[-50, 50]$ range. Cases where it exceeds 100% (i.e., enhanced-planner solutions are more than double in length) are rare. We leave a better study of plan quality as future work.

7 Conclusion and Future Work

The problem of re-configuring a system, such that it subsequently functions as desired, can be formalized as a planning problem with temporal goals and uncontrollable events. This paper introduces an incremental search algorithm and a heuristic based on action relevance, two generally applicable planning enhancements. We apply these to planning with temporal goals and uncontrollable events. An existing planner designed for this model, that can achieve an impressive speed-up due to an online learning method, is used as a benchmark in experiments. Our enhancements further improve the speed considerably.

In the future, we plan to add goal ordering heuristics to the incremental search approach, aiming at further speed-up and better solution quality. The incremental algorithm and the relevance heuristic can easily be adapted to classical planning. We plan to use them in combination with existing heuristics, e.g., by using the relevance heuristic as a secondary ordering criterion, to break ties in the main heuristic.

References

[Amir and Engelhardt, 2003] E. Amir and B. Engelhardt. Factored Planning. In *IJCAI-03*, pages 929–935, 2003.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artif. Intelligence*, 16:123–191, 2000.

[Baier *et al.*, 2007] J. Baier, F. Bacchus, and S. McIlraith. A Heur. Search Approach to Planning with Temporally Extended Preferences. In *IJCAI-07*, pages 1808–1815, 2007.

[Barbeau *et al.*, 1998] M. Barbeau, F. Kabanza, and R. St-Denis. A Method for the Synthesis of Controllers to Handle Safety, Liveness, and Real-Time Constraints. *IEEE Transact. on Automatic Control*, 43(11):1453–1559, 1998.

[Brafman and Domshlak, 2008] R. I. Brafman and C. Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *ICAPS-08*, pages 28–35, 2008.

[Ciré and Botea, 2008] A. A. Ciré and A. Botea. Learning in Planning with Temporally Extended Goals and Uncontrollable Events. In *ECAI-08*, pages 578–582, 2008.

[Edelkamp *et al.*, 2006] S. Edelkamp, S. Jabbar, and M. Nazih. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *Booklet of IPC-5*, 2006.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14:253–302, 2001.

[Hsu *et al.*, 2006] C. W. Hsu, B. W. Wah, R. Huang, and Y. X. Chen. Handling Soft Constraints and Preferences in SGPlan. In *ICAPS Workshop on Preferences and Soft Constraints in Planning*, pages 54–57, 2006.

[Kelareva *et al.*, 2007] E. Kelareva, O. Buffet, J. Huang, and S. Thiébaux. Factored Planning Using Decomposition Trees. In *IJCAI-07*, pages 1942–1947, 2007.

[Koehler, 1998] J. Koehler. Solving Complex Planning Tasks Through Extraction of Subproblems. In *AIPS-98*, pages 62–69, 1998.

[Mittal and Frayman, 1989] Sanjay Mittal and Felix Frayman. Towards a generic model of configuration tasks. In *IJCAI-89*, pages 1395–1401, 1989.

[Sabin and Weigel, 1998] Daniel Sabin and Rainer Weigel. Product configuration frameworks - a survey. *IEEE Intelligent Systems*, 17:42–49, 1998.

[Vidal, 2004] V. Vidal. A Lookahead Strategy for Heuristic Search Planning. In *ICAPS-04*, pages 150–159, 2004.