Heuristics for HTN Planning As Satisfiability *

Amol D. Mali
Dept. of Electr. Engg. & Computer Science,
University of Wisconsin, Milwaukee, WI 53211, USA.
mali@miller.cs.uwm.edu, Phone: 414 - 906 - 8942

Abstract

Classical planning is the problem of synthesizing a sequence of actions to reach the goal state starting from the initial state. Hierarchical task networks (HTNs) have been used as a guidance for solving planning problems where tasks are decomposed into subtasks. Recently, both action-based planning as well as HTN-based planning have been cast as SAT. The performance of planning as SAT has been hitherto remarkably improved by using a variety of techniques. However none of these techniques is sensitive to the structure of domain specific knowledge. To bridge this gap, we develop and evaluate several heuristics that are sensitive to the structure of the given domain knowledge (in the form of HTNs), to generate propositional encodings of HTN planning. The resulting encodings are smaller and easier to solve on most of the problems. Given that the current SAT solvers can handle a limited number of variables (10,000) and clauses (100,000) in real time, such heuristics sensitive to the structure of the domain knowledge are important.

1 Introduction

Classical planning is the problem of synthesizing an executable sequence of actions to reach the partially specified goal state starting from the completely specified initial state, assuming deterministic actions and static and perfectly observable environment. An action has pre-conditions which are atoms (positive literals) and effects which are atoms (positive and negative literals), classified into add effects and delete effects. For example, to execute the action $load(A, R_1, London)$ (loading package A into rocket R_1 at London), the pre-conditions $at(A, London), has\_fuel(R_1)$ and $at(R_1, London)$ must be true and after this action is executed, $in(A, R_1)$ will become true (add effect) and $at(A, London)$ will become false (delete effect), assuming that the action descriptions are defined in such a way that a package that is loaded is no longer in the city.

I thank Subbarao Kambhampati for comments on the previous version of this paper.

The traditional classical planners (also known as "split & prune" planners) did refinement search [Kambhampati 97], starting with an empty plan and refining it by adding more constraints like actions and orderings between these. Based on whether the search is conducted in the space of world states or a space of partial plans (sets of constraints about orderings and causal sources of truths of pre-conditions of actions), the planners are broadly classified as "state space" planners and "partial order" planners. More encouraging results were obtained by casting planning as propositional satisfiability [Kautz & Selman 96] and [Kautz & Selman 99]. The general idea of this paradigm is to construct a disjunctive structure that contains all action sequences of length k, some of which may be plans. The problem of checking if there exists a plan is posed as satisfiability testing. The SAT instance (encoding of the planning problem) contains constraints that must hold for any specific sequence to be a solution. The encoding is thus specified in such a way that it has a model if and only if there exists a provably correct plan of k steps. If no model is found, a new encoding is generated by increasing the value of k.

In many domains, human experts can share their knowledge with the planners by specifying decomposition strategies for complex tasks, in the form of schemas that contain constraints that must be satisfied for the tasks to be fulfilled. Planners that use such task reduction schemas are known as "hierarchical task network planners" [Wilkins 88][Erol 95] and [Currie & Tate 85]. For example, the task $Transport(A, London, Paris, R_1)$ of transporting the package A from $London$ to $Paris$ with rocket R_1 may be decomposed by specifying the set of constraints (reduction schema) $\{p_1 : load(A, R_1, London), p_2 : unload(A, R_1, Paris), p_3 : fly(R_1, London, Paris), p_1 \prec p_3\}$, where p_1, p_2 and p_3 are steps whose action bindings are specified. Note that some constraints necessary for the fulfillment of a task may not be specified in the schemas, e.g. $p_3 \prec p_2$

here. An HTN planning problem is solved by decomposing multiple tasks and resolving interactions between the primitive (executable) actions from these tasks till a goal achieving action sequence is found. HTN planners have been used in several fielded applications including beer factory production line scheduling and military operations planning [Wilkins 88]. HTN planning can be viewed as an "augmentation" of action-based planning, where the task reduction schemas provide an implicit grammar of legal ("desired") solutions or user intent [Kambhampati et al. 98]. User intent (certain preferences about plan generation) is captured in the reduction schema, in the form of constraints, so that only those plans that respect these constraints are generated. Thus plans generated by HTN planners, in addition to being goal achieving, also have a parse in terms of the task reduction schemas. HTNs are used in planning with the motivation of reducing the combinatorics in action-based planning and/or modeling the domain in a particular style of decomposition and plan generation.

Hitherto, the performance of planning as satisfiability has been improved either using a **smaller encoding** like state space (based on state space planning) [Ernst et al 97] and/or propagating **domain specific knowledge** [Mali 99(a)] and/or using **logical inference** like unit propagation (simplifying $(\alpha \wedge (\alpha \Rightarrow \beta))$ to $(\alpha \wedge \beta)$) [Kautz & Selman 99] (especially using unit clauses in initial and goal states) and/or using different action representations (e.g. splitting $move(A, B, C)$ action as $(move\_arg1\_A \wedge move\_arg2\_B \wedge move\_arg3\_C)$) or using different frame axioms for explaining the change in the world state (explanatory rather than classical, see sect. 2.2) [Ernst et al 97] or eliminating actions as in the state-based encodings [Kautz et al 96] and/or using better SAT solvers like satz-rand [Kautz & Selman 99] that do not get stuck in undesirable regions of search space. All these approaches, though remarkable advances, are not sensitive to the structure of the domain specific knowledge. Our work is about efficiently representing the domain knowledge given in the form of HTNs, exploiting their structure, so that the resulting encodings of HTN planning can be simplified to a greater extent, respecting the grammar of legal solutions. We propose several heuristics to achieve such representations.

Encodings of action-based planning [Kautz et al 96] disjunctively represent the potential $step \rightarrow action$ bindings. For example, the clauses $((p_1 = o_1) \vee (p_1 = o_2) \vee (p_1 = o_3))$ and $((p_2 = o_1) \vee (p_2 = o_2) \vee (p_2 = o_3))$ compactly encode 9 action sequences, each containing 2 actions, where p_1, p_2 are steps and o_1, o_2 and o_3 are actions. Assuming that the task reduction schemas

are not recursive, [Mali 99(a)] set up HTN planning encodings (we refer to these as HTN encodings for brevity) by bounding the number of such schemas required to solve a problem and constraining the action-based encodings (state space and causal) in [Kautz et al. 96], with the task reduction schemas. The HTN encodings contain, as a disjunction, both the potential non-primitive step $\rightarrow$ non-primitive task bindings and the potential $step \rightarrow action$ bindings. In their causal HTN encodings (sect. 2.4), the non-primitive step $\rightarrow$ non-primitive task bindings, along with the actions that occur in the reduction schemas, are used to decide the disjunctive $step \rightarrow action$ binding. Since a step is not bound to an action outside the task reduction schemas, one does not have to consider all ground actions in the disjunctive action binding of a step. [Mali 99(a)] propagate this smaller disjunctive binding to demonstrate a performance improvement over action-based encodings (that do not contain the constraints from HTNs). We report several heuristics for improving their strategy of using the task reduction schemas to decide the disjunctive $step \rightarrow action$ binding, in polynomial time.

The heuristics are based on a purely quantitative analysis like the number of common primitive (directly executable) actions among the task reduction schemas and/or the number of their common pre-conditions and/or common add effects and/or common delete effects. The task reduction schemas in most planning domains have such common actions or contain actions having some common pre-conditions and/or some common add effects and/or some common delete effects. For example, though the actions $load(A, R1, London)$ and $load(A, R2, London)$ load the same package into different rockets, they both have the common precondition $at(A, London)$ and the common delete effect $at(A, London)$. Different heuristics exploit different common features to reduce the domains of certain key variables (note that constraint propagation cannot achieve this kind of reduction) in the encodings of HTN planning and consequently, the domains of other variables shrink as well, when the reduction in the domains of the key variables (like the reduction in the number of potential action bindings (domain) of a step (variable)) is propagated (by additional reasoning which heuristics do not carry out, there is another code for doing it). Thus the encoding sizes go down, both because of the heuristics and the constraint propagation that succeeds them. The heuristics do not use the initial and goal states of the problem, generating a problem independent output for the given task reduction schemas. The heuristics are however dependent on the structure in which the domain knowledge is specified.

This paper makes the following contributions.

- We develop several polynomial time domain independent pre-processing heuristics to generate the disjunctive *step* → *action* binding in the HTN encodings. The heuristics reduce the domains of some key variables, increasing the possibilities of simplification by propagation of this domain reduction. The heuristics do not affect the correctness and the number of plans found.

- The heuristics are sensitive to the structure of the domain knowledge. Since the heuristics do not need to know the initial and goal state of the problem, they generate a problem independent result for given HTNs.

- An empirical evaluation of the heuristics which shows that the encodings generated using each of them are indeed smaller and faster to solve on most of the problems and that for each of the solved problems, there was at least one heuristic that yielded an encoding that was the fastest to solve.

Since current SAT solvers handle a limited number of variables (10^4) and clauses (10^6) in real time, it is important to find techniques to simplify the encodings so that their sizes are within these limits. Thus heuristics exploiting the structure in the domain knowledge are important.

The paper is organized as follows. In section 2, we explain the basics of action-based encodings of [Kautz et al 96] and the causal HTN encoding [Mali 99(a)]. In section 3, we describe 5 heuristics for generating the disjunctive *step* → *action* binding. In section 4, we discuss the empirical results. We survey heuristic and constraint-based approaches to plan synthesis in section 5. The conclusions are reported in section 6.

2 Background

In this section, we provide the necessary basics of the action-based encodings of [Kautz et al 96] and the causal HTN encoding of [Mali 99(a)].

2.1 Notation

p_i denotes a plan step. o_j denotes a STRIPS style ground action. ϕ denotes the null action (no-op) which has no pre-conditions and no effects. ϕ is used to generate plans requiring fewer actions or task reduction schemas than the chosen bound. $(p_i = o_j)$ denotes the step→action binding.

$p_i \xrightarrow{f} p_j$ denotes a causal link. $p_i \prec p_j$ denotes that p_i precedes p_j. N_i denotes a ground non-primitive task. s_i denotes a non-primitive step that is bound

to a non-primitive task. $(s_i = N_j)$ denotes the non-primitive step → non-primitive task binding. r_{ij} denotes jth reduction schema of N_i. A reduction schema may contain actions, non-primitive tasks, causal links and orderings between these, as in [Erol 95]. When we refer to task, we always mean non-primitive task. If a step is bound to a task, it is always a non-primitive step.

2.2 Action-based state space encoding

To prove that a sequence of actions is a plan, state space methods essentially try to progress the initial state I (or regress the goal state G) through the sequence to see if the goal state (or initial state) is reached. The following constraints are represented in the explanatory frame axiom-based state space encoding of [Kautz et al 96].

1. The initial state is true at time 0 and the goal must be true at time k. **2.** Conflicting actions (one action deleting the pre-condition or effect of another or needing negation of pre-condition of another) cannot occur at the same time step. **3.** The "explanatory frame axioms" state that if the truth of a fluent changes over the interval $[t, t+1]$, some action changing that must occur at t. **4.** If an action occurs at time t, its pre-conditions are true at t and effects are true at $(t+1)$.

2.3 Action-based causal encoding

The plan space (causal) encodings (based on partial order planning) are based on the ideas of proving the correctness of a plan using causal reasoning about the establishment and preservation of goals and the preconditions of individual actions. The following constraints are represented in the causal encoding of [Kautz et al 96].

1. Each step in the encoding is bound to a single action or the no-op. This is stated as a disjunctive *step* → *action* binding and mutual exclusion constraints. **2.** The conditions true or false in the initial state are the effects of step I and the conditions in the goal state are the preconditions of step F. I has no pre-conditions and it is always the first step in a plan and F has no effects and it is always the last step in a plan. **3.** A step inherits the preconditions and effects of its action binding. **4.** The only way a step can add, delete or need a condition is if the condition is added, deleted or needed (respectively) by its action binding. **5.** Each precondition of each step must have a causal link supporting it. **6.** The contributor step of a causal link precedes the consumer step, and if a step is bound to an action that deletes the condition supported by the causal link (threat), that step either precedes the

contributor or succeeds the consumer. **7.** The $\prec$ relation is irreflexive, asymmetric and transitive.

2.4 Basics of HTN Encodings

The propositional encodings for HTN planning [Mali 99(a)] are developed by constraining the action-based encodings of [Kautz et al 96], such that their satisfying models also conform to the grammar of solutions of interest to a user, specified by the task reduction schemas. Since most implemented HTN planners [Wilkins 88] share a lot of structure of the partial order planners, they constrain the causal encoding in [Kautz et al. 96] to develop the encodings for HTN planning. We use their top-down causal HTN encoding in the further discussion and empirical evaluation. The key constraints that appear in this encoding are described below. The complete set of constraints is given in [Mali 99(b)].

1. If a step is bound to a non-null task, it adds the add effects of the task. **2.** If a step is bound to a task, all constraints in some reduction of that task must be satisfied. **3.** A step is bound to a single task. This is stated as a disjunction of the potential step $\rightarrow$ task bindings and pairs of mutually exclusive bindings. **4.** Every goal condition is added by some task.

We reproduce some notation and preliminaries from [Mali 99(a)] next. M_i denotes the maximum number of actions in reduction schema of the task N_i and $M = max(\{M_i \mid i \in [1, m]\})$, m being the number of tasks given. K denotes the number of non-primitive steps used in the HTN encoding. The total number of steps in an HTN encoding are T, where $T = M.K$, since the reduction schemas are not recursive and the steps are not bound to actions outside the reduction schemas. This can be viewed as allocating M steps ranging from $p_{(i-1).M}$ to $p_{(i.M)-1}$ to each non-primitive step s_i (M is computed automatically by examining the reduction schemas). The action bindings of the T steps are decided by the tasks chosen and the actions in the reduction schemas chosen to fulfill the tasks.

2.5 Disjunctive *step* → *action* binding

Consider two tasks N_1 and N_2 such that N_1 is fulfilled by the reduction schema r_{11} which is $\{ o_1, o_2, o_1 \prec o_2, N_2 \}$ and N_2 has two reduction schemas r_{21} and r_{22}. Let r_{21} be $\{ o_3, o_4, o_5, o_3 \xrightarrow{f} o_4 \}$ and r_{22} be $\{ o_5, o_1, o_5 \prec o_1 \}$. The maximum number of actions (M) that occur in a reduction schema is 3 (in r_{21}). If $K = 2$, $T = 6$. Let p_1, p_2 and p_3 be the steps allocated to s_1 and p_4, p_5 and p_6 be the steps allocated to s_2. Since the action bindings of steps depend on the non-primitive step $\rightarrow$ task bindings, $((s_2 = N_2) \Rightarrow (A \vee B))$, such that $(A \Rightarrow ((p_4 = o_3) \wedge (p_5 = o_4) \wedge (p_6 = o_5)))$
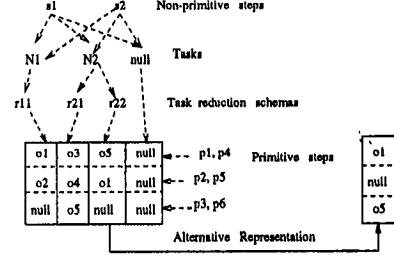

Figure 1: The potential action bindings of steps in causal HTN encoding

and $(B \Rightarrow ((p_4 = o_5) \wedge (p_5 = o_1) \wedge (p_6 = \phi)))$. Similarly, formulae can be written for the cases $(s_1 = N_2)$, $(s_2 = N_1)$, $(s_1 = N_1)$, $(s_1 = \phi)$ and $(s_2 = \phi)$.

[Mali 99(a)] bind the steps to actions in a naive style, based on the order in which the actions appear in the data structure containing a reduction schema, such that steps with lower indices are bound to actions that occur earlier in the data structure. Thus if the actions from the reduction schema r_{11} appear in the order o_1, o_2 in the data structure, they bind p_1 to o_1, p_2 to o_2 and p_3 to no-op and state that the variable $(s_1 = N_1)$ implies a conjunction of these bindings. The actions that occur in the disjunctive *step* → *action* binding for a step can be put together in a set. We denote the set of potential action bindings of a step p_i by $S(p_i)$.

Assuming that the order in which the actions are written in the descriptions of the reduction schemas above are same as the order in which they appear in the data structure, the $S(p_i)$s generated by the [Mali 99(a)] approach are - $\{o_1, o_3, o_5, \phi\}$ (for p_1 and p_4), $\{o_2, o_4, o_1, \phi\}$ (for p_2 and p_5) and $\{o_5, \phi\}$ (for p_3 and p_6), also shown in the regular representation in Fig. 1. Stated in terms of clauses, these are - $((p_1 = o_1) \vee (p_1 = o_3) \vee (p_1 = o_5) \vee (p_1 = \phi))$, $((p_4 = o_1) \vee (p_4 = o_3) \vee (p_4 = o_5) \vee (p_4 = \phi))$, $((p_2 = o_1) \vee (p_2 = o_2) \vee (p_2 = o_4) \vee (p_2 = \phi))$, $((p_5 = o_1) \vee (p_5 = o_2) \vee (p_5 = o_4) \vee (p_5 = \phi))$ and so on.

An action-based causal encoding of k steps contains $O(k^2 \mid U \mid)$ causal link variables, since a causal link is generated for each of the $\mid U \mid$ pre-conditions in the domain, considering each of the k steps as a potential contributor and each of the remaining $(k - 1)$ steps as a potential consumer. The encoding also contains $O(k^3 \mid U \mid)$ clauses for resolving threats, since each of the $O(k^2 \mid U \mid)$ potential causal links may be potentially threatened by any of the remaining $(k-2)$ steps. $S(p_i)$ is very important because it can be used to decide whether p_i can add, delete or need a condition. If no element of $S(p_i)$ adds or deletes or needs a condition, p_i will indeed never add or delete or need this

condition respectively.

The causal link variable $p_i \xrightarrow{f} p_j$ need not be generated if (i) $S(p_i)$ does not contain any action that adds f or (ii) $S(p_j)$ does not contain any action that needs f. Similarly, the threat resolution clause $((p_i \xrightarrow{f} p_j) \wedge Dels(p_s, f)) \Rightarrow ((p_s \prec p_i) \vee (p_j \prec p_s))$, need not be generated if (i) $S(p_i)$ or $S(p_j)$ show that $p_i \xrightarrow{f} p_j$ will always be false or (ii) no element of $S(p_s)$ deletes f. Using $S(p_i)$, one can not only reduce the number of causal link variables and threat resolution clauses, but also the number of variables like $Adds(p_i, f), Needs(p_i, f)$ and $Deletes(p_i, f)$ and mutual exclusion clauses like $\neg((p_i = o_j) \wedge (p_i = o_q))$. The asymptotic number of such exclusions is $O(T \mid O \mid^2)$, O being the set of ground actions. [Mali 99(a)] have reported empirical results showing reductions in the sizes and solving times, yielded by this $S(p_i)$ based reasoning.

3 Heuristics

The five heuristics we develop in this section are variants of the strategy of computing $S(p_i)$ of [Mali 99(a)]. Consider the example in Fig. 1. If the positions of o_5, o_1 and ϕ (null) from r_{22} are changed as shown in the alternative representation, for each $p_i, i \in [1, 6]$, $S(p_i)$ in the new representation is a subset of $S(p_i)$ in original representation. For example, the disjunctive step→action binding for steps p_1, p_4 changes, that is, $((p_1 = o_1) \vee (p_1 = o_3) \vee (p_1 = \phi))$, in the alternative representation. The new representation will lead to a provably smaller encoding than the original. Note that we have not changed the contents of the task reduction schemas, while coming up with the new alternative.

We give below a generalized template for computing $S(p_i)$ and then describe the heuristics. $S(p_{(i+M)})$ is same as $S(p_i)$, as can also be seen from Fig. 1. Thus to compute such sets for $K.M$ steps, it is enough to compute these sets for the M steps $p_1, ..., p_M$. R denotes the sum of the number of reduction schemas of all the given tasks. $C(r_{ij})$ denotes a copy of the set of actions in reduction schema r_{ij} (we assume that there are symbols to distinguish between multiple occurrences of an action). Each $S(p_i)$ contains the no-op ϕ. The computation in the template terminates in $O(MRh)$ time, where h is the time required for applying the heuristic on an iteration. At the time of termination, all $C(r_{ij})$ are empty.

1. **For** each step p_i, $i \in [1, M]$, $S(p_i) = \{\}$
2. **For** each task $N_j, j \in [1, m]$,
3. **For** each reduction schema r_{jq} of N_j,
Choose an action $o_s \in C(r_{jq})$ using

the **chosen heuristic** and do
$\{ S(p_i) \leftarrow (S(p_i) \cup \{o_s\}),$
$C(r_{jq}) \leftarrow (C(r_{jq}) - \{o_s\}),$ Go to 3.$\}$

If a, d, p are the respectively the maximum number of add effects, delete effects and preconditions an action may have, the complexities of applying the heuristics 2, 3, 4 and 5 to generate $S(p_i)$ for M steps are $O((MRa)^2)$, $O((MRd)^2)$, $O((MRp)^2)$ and $O((MRs)^2)$ respectively, where s is the maximum of the individual sums of the number of preconditions and add effects of individual actions.

Note that minimizing $\mid S(p_i) \mid$ will not necessarily lead to the smallest encoding, since the potential effects and preconditions of steps decide the potential causal links and threats. The orderings and causal links from the reduction schemas contribute to the encoding size as well. Computing the $S(p_i)$ sets that will lead to the smallest encoding is combinatorially hard (that is why we propose a variety of heuristics below). We do not claim that small encodings are always easier to solve.

The different heuristics below attempt to reduce the numbers of different types of clauses and variables. The empirical results in Fig. 2 and 3 show that indeed, the encodings generated using different heuristics have significantly different sizes or solving times, especially on larger problems.

1. **Least increase in size of $S(p_i)$** While choosing the action o_s to include in the current $S(p_i)$, an action that is already an element of $S(p_i)$ is preferred to an action that is not. If it is necessary to include a new action, it is chosen arbitrarily. Since there are M steps whose $S(p_i)$ needs to be computed and the maximum value of $\mid S(p_i) \mid$ is R, and $O(MR)$ comparisons are needed to add an action to $S(p_i)$, the complexity of computation in the template is $O((MR)^2)$. The motivation for introducing this heuristic is to reduce all types of variables and clauses that contain primitive steps or actions. The results (Fig. 2) show that this heuristic lead to the lowest average encoding solving time.

2. **Least increase in the size of the set of potential add effects** Let $A(S(p_i))$ denote the union of the add effects of actions in current $S(p_i)$. While choosing the action o_s to include in $S(p_i)$, the size of the intersection of the set of add effects of o_s and $A(S(p_i))$ is considered. o_s is included in $S(p_i)$ if the size of the intersection is the least. If there is a tie, choice of o_s is made arbitrarily. The motivation for introducing this heuristic is to reduce the number of causal links, threats and also the variables of type

$Adds(p_i, f), p_i \prec p_j$.

3. Least increase in the size of the set of potential delete effects Let $D(S(p_i))$ denote the union of the delete effects of actions in current $S(p_i)$. This works similar to heuristic 2, however here the intersection of the set of delete effects of o_s and $D(S(p_i))$ is considered. The motivation for introducing this heuristic is to reduce number of threats, step ordering variables (needed to resolve threats), step orderings following from these by transitivity and $Deletes(p_i, f)$ type variables.

4. Least increase in the size of the set of potential preconditions Let $P(S(p_i))$ denote the union of the preconditions of actions in current $S(p_i)$. This works similar to heuristic 2, however here the intersection of the set of preconditions of o_s and $P(S(p_i))$ is considered. The motivation for introducing this heuristic is to reduce the number of causal links, threats and variables of type $Needs(p_i, f)$ and $p_i \prec p_j$.

5. Least increase in the sum of the sizes of $A(S(p_i))$ and $P(S(p_i))$ While choosing the action o_s to include in current $S(p_i)$, the sum of the sizes of the intersections in heuristics 2 and 4 is considered. The motivation for introducing this heuristic is to reduce the number of causal links, threats, step ordering variables and variables of type $Adds(p_i, f)$ and $Needs(p_i, f)$. The results (Fig. 2 and 3) show that this heuristic always lead to encodings with lowest number of clauses and variables.

4 Empirical Evaluation

To test the effectiveness of our heuristics, we conducted an empirical evaluation on several benchmark domains. The descriptions of the benchmark domains used and the "satz" systematic SAT solver we used are available at ftp://ftp.cs.yale.edu/pub/mcdermott/domains/ and http://aida.intellektik.informatik.th-darmstadt.de / ~ hoos/SATLIB/ respectively. Tsp is the traveling salesperson domain. The task reduction schemas were created by putting together actions for traveling (Tsp), loading and unloading packages and flying planes (transportation logistics), decorating, installation and construction (house building) and barking, debarking and sailing (Ferry). The description of the house building domain can be found on the home page of AIAI, Edinburgh, UK. The plans in the Tsp and Ferry domains were purely serial. Thus the number of actions in plans in these domains is same as the plan lengths. (Notation in Fig. 2, 3 - V, C, T de-

note the number of variables, clauses in and the times needed to solve the encodings respectively. Times of the solved encodings are in CPU seconds. A "*" indicates that the encoding could not solved within 5 minutes. A "-" denotes that the encoding was too large to store. **LA, LAE, LDE** and **LPC** are the **least increase in the set of potential** action bindings, add effects, delete effects and preconditions **heuristics** respectively. **LPCA** is the least increase in the sum of the sizes of the sets of potential add effects and pre-conditions heuristic. **Caus.** and **Stat.** are the causal HTN encoding based on naive $S(p_i)$ computation strategy and the state space HTN encoding from [Mali 99(a)] respectively.)

The heuristics are integrated with the code that generates the encodings. Task reduction schemas not required for solving the problems were not used in generating the encodings. It is very common in HTN planning to use only the relevant task reduction schemas, e.g. [Smith & Nau 93] reduced the number of nodes in the game tree (for playing bridge) from 6.01×10^{44} to 1300, by using only the relevant task reduction schemas. The encodings were generated and solved on a Sun Ultra with 128 M RAM. The K chosen was same as the number of reduction schemas required to solve the problems, however this need not be the case. Note that K is a parameter input to the encoding generator and K is not a property of the domain. Like [Ernst et al 97], we do not report the times required to simplify the encodings, since these times were significantly small. When we talk of reduction in encoding sizes, it is a reduction over the causal HTN encoding of [Mali 99(a)] (which is simplified by pre-processing without heuristics), unless stated otherwise. The empirical results are reported in Fig. 2 and 3.

In a k step state space encoding, a link $o_a \xrightarrow{f} o_b$ needs to be represented as $\vee_{i=0}^{k-2} \vee_{j=i+1}^{k-1} (o_a(i) \wedge o_b(j) \wedge (\wedge_{q=i+1}^{j} f(q)))$, where $o_a(i)$ and $o_b(j)$ denote that o_a and o_b occur at times i and j respectively and $f(q)$ denotes that f is true at q. Because of this, the state space encodings were always the largest. Thus we do not evaluate them on larger problems (Fig. 3). Below we discuss experimental results from Fig. 2.

The **LA** heuristic ($| S(p_i) |$ minimization) is likely to yield improvement when task reduction schemas have significant number of actions in common. The schemas from the logistics and ferry domain did have transportation related actions in common and the schemas from house building domain did have construction related actions in common. Thus the first heuristic did yield encodings containing fewer variables and fewer clauses in these domains.

The **LAE** heuristic ($| A(S(p_i)) |$ minimization) is

Domain, # Actions in plan	LA V, C, T	LAE V, C, T	LDE V, C, T	LPC V, C, T	LPCA V, C, T	Caus. V, C, T	Stat. V, C, T
Ferry, 8 K = 2, T = 8	235	211	221	231	211	235	421
	939	823	883	925	823	947	1750
	0.03	0.03	0.03	0.03	0.02	0.05	0.06
Ferry, 15 K = 4, T = 16	827	827	811	811	747	1043	4767
	6359	6359	6267	6267	5671	9239	32368
	0.61	0.59	0.53	0.57	1.2	5.81	*
Tsp, 12 K = 6, T = 12	1189	1189	1189	1189	1189	1189	3289
	6397	6397	6397	6397	6397	6409	17534
	0.96	0.98	0.92	0.96	0.96	0.61	1.08
Logistics, 15 K = 4, T = 16	758	794	794	634	634	990	4930
	5786	6154	6154	3778	3778	8738	32323
	0.4	0.58	0.59	0.29	0.28	0.88	*
Build House, 46 K = 6, T = 48	4371	4413	4317	4239	4239	8463	449893
	112829	112949	112673	112451	112451	124883	8180596
	24.4	28.94	24.19	22.38	22.46	57.87	-

Figure 2: Empirical evaluation of the 5 heuristics on smaller problems.

Domain, # Actions in plan	LDE V, C, T	LPC V, C, T	LPCA V, C, T	Caus.WR V, C, T	Caus. V, C, T
Logistics, 27 K = 7, T = 28	2256	2137	2137	3411	3411
	29682	28604	28604	61154	61147
	5.45	5.03	5.39	18.13	> 45 min
Logistics, 35 K = 9, T = 36	3818	3629	3629	6041	6041
	63704	61616	61616	152597	152588
	18.28	17.36	17.39	148.41	> 15 min
Ferry, 24 K = 6, T = 24	1863	1617	1617	2511	2511
	21039	18453	18453	35799	35793
	3.1	2.26	2.16	10.47	189.34
Ferry, 31 K = 8, T = 32	3219	3219	2995	4875	4875
	48555	48555	44451	99787	99779
	10.63	10.53	7.9	59.99	> 10 min.
Ferry, 39 K = 10, T = 40	5143	5143	4803	7693	7693
	95623	95623	87833	195143	195133
	49.04	48.85	21.27	647.64	> 7 hrs
Ferry, 43 K = 11, T = 44	6306	6306	5899	9375	9375
	128043	128043	117780	260230	260219
	50.08	50.73	34.11	749.38	> 15 min.
Ferry, 55 K = 14, T = 56	10671	10671	10027	15585	15585
	269811	269811	249105	541327	541313
	155.29	155.12	124.17	-	-

Figure 3: Empirical evaluation of the heuristics on larger problems.

likely to yield improvements when actions from different reduction schemas have significant number of common add effects (note that no $S(p_i)$ can contain more than 1 action from the same reduction schema, this is also clear from Fig. 1 and the template). Since this was the case in the ferry and logistics domains (for example, loading (into same vehicle) and unloading of objects (at the same location)), this heuristic yielded improvement in the sizes and solving times. The **LDE** heuristic is likely to yield improvement when actions from different reduction schemas have significant number of common delete effects. On many problems, this heuristic was not the best one because the the number of common add effects dominated the number of common delete effects.

The **LPC** heuristic yielded better results when actions from different reduction schemas had lot of common preconditions, as in some problems from the ferry, house building and logistics domains. The **LPCA** heuristic yielded better results when actions from different reduction schemas have significant number of common preconditions and/or add effects. Reduction schemas fom the Tsp domain did not have any actions in common and the actions did not have any common preconditions or add effects or delete effects. Thus it is not a surprise that all heuristics lead to the same encoding. The maximum reductions in the number of variables, clauses and solving times that we obtained on smaller problems (Fig. 2) were 50%, 57% and 90% respectively. Note that there are several other domains like mprime, mystery, molgen-strips and monkey (descriptions available at the Yale URL cited earlier) where the actions have common pre-conditions or add effects or delete effects and we expect the heuristics to yield better results in the hierarchical versions of these domains as well.

No single heuristic lead to the fewest number of variables or fewest clauses or the smallest solution times on all the problems. Different heuristics are sensitive to different properties of actions in the reduction schemas and thus affect different parts of the encodings. Note that we did not create artificial domains that had the distinguishing properties that the heuristics exploit. The performance difference between our heuristics is likely to increase if they are evaluated on such domains.

The LPCA heuristic always lead to encodings with the fewest variables and clauses due to the more information it used (add effects as well as pre-conditions). Other heuristics use either only the information about action names or only the pre-conditions or only the add effects or only the delete effects and because of this narrow focus, they did not lead to $S(p_i)$s that lead to the fewer number of clauses and variables than those

in the one lead to by LPCA. Based on the results in Fig. 2, the heuristics can be ranked as LPCA, LPC, LDE, LA, LAE (as per an increasing order of the average number of variables), as LPCA, LPC, LDE, LA, LAE (as per an increasing order of the average number of clauses) and as LPC, LPCA, LDE, LA, LAE (as per an increasing order of the average number of variables). Because of this and the fact that the sizes of the encodings generated by LA, LAE and LDE were close on larger problems as well, we did not evaluate LA and LAE on larger problems.

None of the larger problems from the chosen domains (Fig. 3) could be solved within half an hour. Thus we decided to add more information to the encodings in the form of the number of occurrences of each non-primitive task needed to solve the problems, as well as the non-primitive step→non-primitive task bindings, for example, $(s_1 = N_1) \wedge (s_2 = N_2) \wedge (s_3 = N_3)$. The experimental results obtained with this additional information are shown in Fig. 3 (satz was used to solve all these encodings). Caus. in Fig. 3 is the causal HTN encoding from [Mali 99(a)] (that was simplified, but which did not use any heuristics to better compute $S(p_i)$). Caus. WR in Fig. 3 is their encoding with the additional information in the form of $(s_i = N_j)$ type bindings. Thus the number of variables in the Caus. and Caus. WR encodings is the same and the number of clauses differ by a very small amount.

The maximum reductions in the number of variables, clauses and solving times we obtained on larger problems were 40%, 60% and 95% respectively (Fig. 3). These are reductions over the corresponding values for the Caus. WR encoding (to compare encodings with the same additional information). Note that though the additional information helped in solving the larger problems, the heuristics still had a significant impact on the solving times, as shown by these reductions. On these larger problems, the LPCA heuristic always lead to encodings with the fewest number of variables and clauses, as well as the lowest solving times.

Besides measuring the total number of variables and clauses in the encodings, we also measured the number of different types of variables (e.g. $Adds(p_i, f), Dels(p_i, f), p_i \prec p_j$, causal links and $Needs(p_i, f)$ etc.) and clauses (e.g. those needed for threat resolution, mutual exclusion and enforcing transitivity). We conducted this dissection of the total sizes to find if the heuristics indeed reduced the number of variables and clauses that they were expected to. An interesting trend revealed by this study is that though the naive causal HTN encodings (for which $S(p_i)$ is same as the set of all ground actions from all given task reduction schemas) contain more causal link vari-

ables ($O(T^2 * \mid U \mid)$) than the number of precedence variables ($O(T^2)$), some of the simplified causal HTN encodings of smaller as well as larger problems (pre-processed using our heuristics or [Mali 99(a), 99(b)] strategy) had more precedence variables than causal link variables. This is because the number of precedence variables increases due to the need to enforce transitivity (that is, if $p_i \prec p_j$ and $p_j \prec p_q$ are created, $p_i \prec p_q$ should be created as well). Similarly, though the number of threats resolution clauses in naive causal HTN encodings ($O(T^3 * \mid U \mid)$) is higher than the number of transitivity axioms ($O(T^3)$), the reverse holds for the simplified encodings of some of the smaller and larger problems.

We also evaluated the heuristics on problems from other domains like moving trains to different tracks (Meet pass) and putting objects into a briefcase, taking them out and moving the briefcase (Briefcase domain), where the heuristics did yield reductions in the encoding sizes and solving times. We also evaluated 3 more heuristics - 1. Least increase in size of $(A(S(p_i)) \cup D(S(p_i)))$ (LADE) 2. Least increase in the size of $(D(S(p_i)) \cup P(S(p_i)))$ (LDPC) and 3. Least increase in the size of $(D(S(p_i)) \cup P(S(p_i)) \cup A(S(p_i)))$ (LADEPC). However these 3 heuristics did not yield significantly higher reductions in the sizes of the encodings or their solving times than those achieved by the five previously discussed heuristics.

In many cases (as can be seen from the sizes in Fig. 2 and 3), different heuristics lead to the same encoding because of the relationships between the criteria used by different heuristics. It is rare for actions in the planning domains to have (i) same add effects and different pre-conditions and/or delete effects, or (ii) same delete effects and different pre-conditions and/or add effects or (iii) same pre-conditions and different add and/or delete effects. Thus $S(p_i)s$ computed by different heuristics may be the same, leading to the same encodings.

Several more heuristics for computing the $S(p_i)$ sets can be developed. Note that when computing $S(p_i)$, interactions of actions already included in it with the actions in the already computed $S(p_j), j \in [1, i-1]$ are not taken into account. A heuristic that adds an action to $S(p_i)$ such that the sum of the sizes of the respective intersections of $A(S(p_j)), j \in [1, i-1]$ with the $P(S(p_i))$ resulting from the inclusion of the action is minimum, better models the requirement that there should not be too many steps that make a pre-condition true. This is a useful heuristic since this requirement also reduces the number of causal link variables. The time required to generate $S(p_i)$ sets using such heuristics is larger than the ones we evaluated,

but it is still low order polynomial. The strategy of randomly breaking ties while computing $S(p_i)$ can be replaced by finer criteria that take into account more features of the reduction schemas. The distribution of actions among the reduction schemas matters. For example, the heuristics perform better when n reduction schemas contain an action, than when a single reduction schema containing n occurrences of the action. Heuristics thus need to be sensitive to more features of reduction schemas. This opens up directions for future work on developing such more sensitive heuristics.

[Stone et al 94] have stressed the need to have different heuristics for controlling search in plan generation in domains with different characteristics. We stress the need to have different structure-sensitive heuristics for pre-processing the knowledge from different domains.

5 Constraint-based Plan Synthesis

Planners in [Blum & Furst 95], [Kautz & Selman 96] and [Kautz & Selman 99] cast planning as some form of constraint satisfaction without domain specific knowledge. [Blum & Furst 95] reported the Graphplan planner that builds a structure called "planning graph" and searches it to extract a plan. Odd numbered levels in the planning graph are action levels and even numbered levels are proposition levels. Zeroth proposition level is the initial state. A proposition that occurs in proposition level i occurs in all proposition levels $j > i$. Similarly, actions in i th action level also occur in all j th action levels, $j > i$. i th proposition level is a union of the $(i-2)$ th proposition level and the add and delete effects of actions at $(i-1)$ th action level. A planning graph compactly encodes several action sequences of a certain length. Graphplan propagates mutual exclusion relations between actions in the same action level. [Kautz & Selman 99] convert the planning graph built by Graphplan into a propositional encoding and solve it.

[Kautz & Selman 98] extend the SAT planning approach to include domain specific state constraints, e.g. state invariants and simplifying assumptions (a loaded truck should move immediately). [Lotem & Nau 00] constrain the planning process in Graphplan [Blum & Furst 95] so that the plans also obey the constraints in the task reduction schemas. The simplification approaches in all the planners above that cast planning as some form of CSP with/without domain knowledge differ from our heuristics because these approaches infer additional constraints (generally using the knowledge of initial and goal state) or control the search online (doing dynamic constraint satisfaction) or they have a different type of domain specific knowledge.

6 Conclusion

The performance of planning as satisfiability has been hitherto remarkably improved by various techniques. These techniques are however not sensitive to the structure of the domain knowledge. Motivated by the question of representing the domain knowledge (given in the form of HTNs) in a way that increases the encoding simplification possibilities, we proposed several polynomial time pre-processing and user intent preserving heuristics to achieve these representations. Note that our heuristics themselves do not perform any logical inference on the domain knowledge and do not need to know the initial and goal states of the problem. We empirically demonstrated that on each of the problems, at least one· heuristic yielded an encoding that was the fastest to solve and on most of the problems, all heuristics yielded improvements in the sizes and solving times.

Our heuristics are based on purely quantitative measures and can be integrated with many other techniques of enhancing SAT/CSP-based HTN planning. The ideas in the heuristics can be adapted to other kinds of planning cast as some form of CSP (like integer linear programming) as well, where the domain specific knowledge is provided as sets of constraints, e.g. plans to be merged. Given that the current SAT solvers can handle a limited number of variables (10^4) and clauses (10^6) in real time, the development of such heuristics sensitive to the structure of the domain knowledge is important.

References

[Blum & Furst 95] Avrim L. Blum and Merrick L. Furst, Fast planning through planning graph analysis, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Montreal, 1995.

[Currie & Tate 85] Currie K and Tate A., O-Plan - Control in the open planning architecture, BSC Expert Systems conference, Cambridge university press, 1985.

[Ernst et al 97] Michael Ernst, Todd Millstein and Daniel Weld, Automatic SAT compilation of planning problems, Proccedings of IJCAI-97.

[Erol 95] Kutluhan Erol, Hierarchical task network planning: Formalization, Analysis and Implementation, Ph.D thesis, Dept. of computer science, Univ. of Maryland, College Park, 1995.

[Kambhampati 97] Subbarao Kambhampati, Refinement planning as a unifying framework of plan synthesis, AI magazine, Summer 1997.

[Kambhampati et al. 98] Subbarao Kambhampati, Amol Mali & Biplav Srivastava, Hybrid planning in partially hierarchical domains, Proceedings of AAAI-98.

[Kautz & Selman 96] Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.

[Kautz et al 96] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, Proc. of Knowledge Representation and Reasoning conference, Cambridge, Boston, 1996.

[Kautz & Selman 98] Henry Kautz and Bart Selman, The role of domain-specific knowledge in the planning as satisfiability framework, Proceedings of the Artificial Intelligence Planning Systems Conference, Pittsburgh, 1998.

[Kautz & Selman 99] Henry Kautz and Bart Selman, Unifying SAT-based and graph-based planning, Proceedings of IJCAI-99, Stockholm.

[Lotem & Nau 00] Amnon Lotem and Dana S. Nau, New advances in GraphHTN: Identifying independent subproblems in large HTN domains, Proceedings of the Artificial Intelligence Planning Systems conference (AIPS), Colorado, 2000.

[Mali 99(a)] Amol Mali, Hierarchical task network planning as satisfiability, Proceedings of European Conference on Planning, Durham, UK, 1999.

[Mali 99(b)] Amol Mali, Hierarchical task network planning as satisfiability, Ph.D thesis, Dept. of computer science & engg., Arizona state university, Tempe, May 1999.

[Smith & Nau 93] Stephen J. J. Smith and Dana Nau, Games: Planning and learning, Papers from the 1993 AAAI Fall symposium.

[Stone et al 94] Peter Stone, Manuela Veloso and Jim Blythe, The need for different domain-independent heuristics, Proceedings of the Artificial Intelligence Planning Systems (AIPS), 1994, 164-169.

[Wilkins 88] David Wilkins, Practical planning: Extending the classical AI planning paradigm, Morgan Kaufmann, 1988.