HyperTensioN and Total-order Forward Decomposition Optimizations

JAAMAS Track

Maurício Cecílio Magnaguagno
PUCRS
Porto Alegre, Brazil
mauricio.magnaguagno@acad.pucrs.br

Felipe Meneguzzi
University of Aberdeen & PUCRS
Aberdeen, United Kingdom
felipe.meneguzzi@abdn.ac.uk

Lavindra de Silva
University of Cambridge
Cambridge, United Kingdom
lavindra.desilva@eng.cam.ac.uk

ABSTRACT

Hierarchical Task Network (HTN) planners generate plans using a decomposition process with extra domain knowledge to guide search towards achieving a task. Domain experts develop such domain knowledge through recipes of how to decompose higher level tasks and under what conditions. By leveraging a three-stage compiler design we can support more language descriptions and preprocessing optimizations to exploit such domain knowledge that when chained can greatly improve runtime efficiency. In this paper we evaluate such optimizations with the HyperTensioN HTN planner: winner of the HTN IPC 2020 total-order track.

KEYWORDS

Hierarchical Task Networks; Planning; Compilation

ACM Reference Format:
Maurício Cecílio Magnaguagno, Felipe Meneguzzi, and Lavindra de Silva. 2026. HyperTensioN and Total-order Forward Decomposition Optimizations: JAAMAS Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/IREH8205

1 INTRODUCTION

Hierarchical planning was developed as a means to allow planning algorithms to incorporate domain knowledge into the search engine using an intuitive formalism [14]. Hierarchical Task Network (HTN) planning is the most widely used formalism for hierarchical planning, having been implemented in a variety of systems with similar input languages [3, 10, 15]. Recent research has re-energized work on HTN planning formalisms and search procedures, leading to a new generation of HTN planners [1, 7–9]. In this paper, we outline key design elements and optimizations of the HyperTensioN planner which works by transforming HTN domains and problems to minimize backtracking, as submitted to the 2020 International Planning Competition (IPC) where it won in the total-order track.[1]

We outline the three-stage design architecture in Section 2, with front, middle and back ends for consuming, transforming and generating planning instances, respectively. Section 3 explores the middle-ends employed in the HTN IPC 2020. Section 4 explores the impact on planning time, before our conclusions in Section 5.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/IREH8205

2 THREE-STAGE DESIGN ARCHITECTURE

HyperTensioN was originally developed to automatically convert classical planning instances to hierarchical planning instances [11]. This required a PDDL [13] parser (front-end) and a (J)SHOP [10] description compiler (back-end). Separating the front-end from the back-end enabled us to generate Ruby code within our implementation of a lifted Total-order Forward Decomposition (TFD) [6, chapter 11] planner, which uses a standard progression search [14]. Our compilation is analogous to JSHOP [10]. Parser and compiler modules use the same Intermediate Representation (IR) to share planning instance data. Middle-end extensions fill gaps between description languages, and optimize descriptions, independent of the target planner, and languages for input and output.

This level of flexibility facilitates developing support for new languages, while remaining compatible with the already available extensions, e.g., the DOT [5] compiler for debugging and the HDDL [7] parser for the IPC. As the project grew, the three-stage compiler and the TFD planner modules split, as shown in Fig. 1. The Hype three-stage compiler controls the pipeline, allowing multiple middle-ends to run, even repeatedly, before compilation into the target representation. The HyperTensioN TFD planner is the final part of this pipeline, to output a plan. Eventually, we extended its core search procedure to a variety of other planning tasks, including search on hybrid symbolic-numeric domains [12]. As a lifted total-order planner, HyperTensioN skips grounding: it avoids instantiating operators and methods based on available objects and preconditions, a process that may be time-consuming and limit the planner to only handle a predefined set of objects.

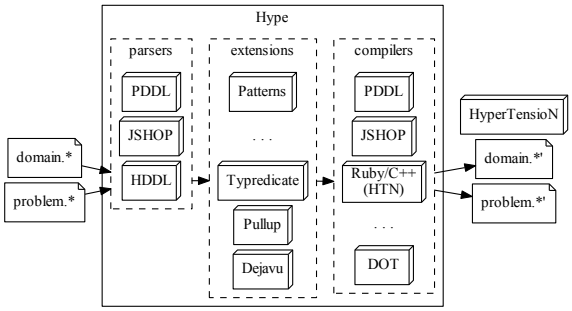

Figure 1: Hype acts as a three-stage compiler before linking with the HyperTensioN planner.

[1] More details available at https://ipc2020.hierarchical-task.net/

3 DOMAIN TRANSFORMATION

To improve planning speed the compiler compresses the state structure by removing rigid predicates and treating them as "constant information". More importantly, we developed extensions to improve the representation to support: (1) better unification exploiting types; (2) early testing of method/action preconditions during decomposition; and (3) a cycle-detection mechanism.

3.1 Typredicate

The Typredicate extension involves constraining the substitutions attempted on variables occurring in predicates, by making better use of constant/parameter types (if the domain expert has not already done so). For example, in the Transport domain, the predicate *(at ?obj – locatable ?l – location)* defines the position of an object. Operator *(drive ?v – vehicle ?l1 ?l2 – location)* uses this predicate to describe the vehicle's position, with *package* and *vehicle* as subtypes of *locatable*. Then, though the *drive* operator never modifies the position of a package (as it checks *(vehicle ?v)*), the *?v* variable (which corresponds to *?obj* in the definition of *at*) may still be *substituted* by constants of type *package* by other operators. Since constants of each subtype are mutually exclusive when the subtypes have the same parent type, we preclude such unnecessary substitutions by specializing *(at ?obj – locatable ?l – location)* into predicates *(at_vehicle_location ?obj – vehicle ?l – location)* and *(at_package_location ?obj – package ?l – location)*, and replacing occurrences of *(at ?obj ?l)* with the appropriate specialized predicates in the domain and problem descriptions.

3.2 Pullup

The Pullup extension implements the main optimization technique that underpins HyperTensioN's performance by "pulling up" preconditions in the hierarchy. It adds a predicate in the precondition (which is a conjunction/set of predicates) of an operator occurring as a method subtask (after variable substitutions) to the precondition of the method if an earlier step in the method does not possibly bring about the predicate, *i.e.*, any solution for the method requires the predicate to hold at the start; we say a predicate is *possibly brought about* (cf. "mentioned" [4]) by a step if there is a predicate asserted by an operator yielded by the step such that the two predicates have the same name (predicate symbol).

3.3 Dejavu

Some domains have methods with direct recursion, where a method includes the same task that it decomposes, or indirect recursion, requiring further decomposition before the search encounters the (same) task. Without "visited" predicates tracked by a domain expert to mark and query visited partial states, such domains can induce an infinite loop for a TFD search procedure. Dejavu transforms the domain by adding "unobservable" primitive tasks to mark and unmark the fact that a particular non-primitive task is being decomposed, and predicates to detect when the task is being recursively (re)attempted. HyperTensioN stores information about such cycles across decomposition branches using an external cache structure, as the state loses the marked information upon backtracking. The cache saves which methods and unifications have been previously explored to avoid repeating decompositions that led to failure.

Table 1: HyperTensioN's multiple configurations with 16GB of RAM and 60s time-out to solve IPC instances.

Domain(instances)		N	T	P	D	TP	PD	TD	TPD
AssemblyHierarchical	(30)	0	0	1	2	1	**3**	2	**3**
Barman-BDI	(20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Blocksworld-GTOHP	(30)	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
Blocksworld-HPDDL	(30)	0	0	0	**28**	0	**28**	**28**	**28**
Childsnack	(30)	27	27	**30**	27	**30**	**30**	27	**30**
Depots	(30)	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**
Elevator-Learned	(147)	**147**	**147**	**147**	**147**	**147**	**147**	**147**	**147**
Entertainment	(12)	3	3	3	5	3	**7**	5	**7**
Factories-simple	(20)	1	1	0	**3**	0	**3**	**3**	**3**
Freecell-Learned	(60)	0	0	0	0	0	0	0	0
Hiking	(30)	0	0	0	**25**	0	**25**	**25**	**25**
Logistics-Learned	(80)	0	0	0	**22**	0	**22**	**22**	**22**
Minecraft-Player	(20)	3	3	3	**5**	3	**5**	**5**	**5**
Minecraft-Regular	(59)	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
Monroe-FO	(20)	6	6	**20**	6	**20**	**20**	6	**20**
Monroe-PO	(20)	0	0	0	0	0	0	0	0
Multiarm-Blocksworld	(74)	2	2	3	**8**	3	**8**	**8**	**8**
Robot	(20)	6	6	6	**14**	6	**14**	**14**	**14**
Rover-GTOHP	(30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
Satellite-GTOHP	(20)	0	0	0	3	0	**20**	3	**20**
Snake	(20)	14	14	15	**20**	15	**20**	**20**	**20**
Towers	(20)	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
Transport	(40)	0	0	0	15	4	25	15	**40**
Woodworking	(30)	5	5	**7**	5	**7**	**7**	5	**7**
Total	**892**	370	370	391	491	395	540	491	**555**

4 OPTIMIZATIONS IMPACT

This section explores the optimizations impact on the IPC total-order set of domains. Table 1 shows the results of our experiments, which comprise the number of instances solved by each configuration for each domain. We highlight the highest values in bold. Two domains have no instances solved within the time restriction: Freecell-Learned and Monroe-PO. Seven domains are unaffected by the extensions, such as Barman-BDI and Elevator-Learned. Typredicate (T) results are similar to No Extensions (N); the same applies for Pullup (P) and Typredicate + Pullup (TP), and Dejavu (D) and Typredicate + Dejavu (TD). Typredicate is only useful for Transport, which when combined with other optimizations were able to solve all instances. Some extensions may add a significant overhead, such as Pullup in Factories-simple, being worse than No extension. In the IPC, HyperTensioN won by a small margin in the total-order track with the TPD configuration. Here we see that D, PD and TPD are the best configurations, while T is equivalent to N.

5 CONCLUSION

HyperTensioN is a planning approach that uses a three-stage compiler designed to support optimizations and transformations. The key to its IPC performance are transformation techniques that replicate domain-knowledge commonly used to speed up search in HTN planners [10], and optimizations used by agent interpreters, *e.g.*, [16]. Our domain transformations enable improvements to the HTN structure for blind Depth First-Search planners using Typredicate and Pullup, while avoiding recomputing parts of complex combinatoric search spaces induced by Transport, Snake, etc. using Dejavu. For more details, see [2].

REFERENCES

[1] Pascal Bercher, Gregor Behnke, Daniel Höller, and Susanne Biundo. 2017. An Admissible HTN Planning Heuristic.. In *IJCAI*. 4384–4390.

[2] Maurício Cecílio Magnaguagno, Felipe Meneguzzi, and Lavindra de Silva. 2025. Hypertension and total-order forward decomposition optimizations. *Autonomous Agents and Multi-Agent Systems* 39, 1 (2025), 1–27.

[3] Lavindra de Silva, Raphaël Lallement, and Rachid Alami. 2015. The HATP hierarchical planner: Formalisation and an initial study of its usability and practicality. In *IROS*. 6465–6472.

[4] Lavindra de Silva, Sebastian Sardina, and Lin Padgham. 2016. Summary Information for Reasoning About Hierarchical Plans. In *ECAI*. 1300–1308.

[5] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. 2001. Graphviz—open source graph drawing tools. In *GD*. Springer, 483–484.

[6] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated planning: theory & practice*. Elsevier, San Francisco.

[7] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. 2020. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 9883–9891.

[8] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. 2018. A Generic Method to Guide HTN Progression Search with Classical Heuristics.. In *ICAPS*.

[9] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. 2020. HTN Planning as Heuristic Progression Search. *JAIR* 67 (2020), 835–880.

[10] Okhtay Ilghami and Dana S Nau. 2003. *A General Approach to Synthesize Problem-Specific Planners*. Technical Report CS-TR-4597. Maryland University, Dept of Computer Science, College Park, Maryland.

[11] Maurício Cecílio Magnaguagno and Felipe Meneguzzi. 2017. Method Composition through Operator Pattern Identification. *KEPS 2017* (2017), 54–61.

[12] Maurício Cecílio Magnaguagno and Felipe Meneguzzi. 2020. Semantic Attachments for HTN Planning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 9933–9940.

[13] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. *PDDL-the planning domain definition language*. Technical Report CVC TR-98-003/DCS TR-1165. Yale Center for Computational Vision and Control.

[14] Dana Nau, Yue Cao, Amnon Lotem, and Hector Muñoz-Avila. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI*. 968–973.

[15] Dana S Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. 2003. SHOP2: An HTN planning system. *JAIR* 20 (2003), 379–404.

[16] John Thangarajah, Lin Padgham, and Michael Winikoff. 2003. Detecting & Avoiding Interference Between Goals in Intelligent Agents.. In *IJCAI*. 721–726.