A comparison of BDI based real-time reasoning and HTN based planning

De Silva, Lavindra; Padgham, Lin

A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning

Lavindra de Silva and **Lin Padgham**

{ldesilva,linpa}@cs.rmit.edu.au

School of Computer Science and Information Technology
RMIT University, Melbourne, Vic., Australia, 3000

Abstract. The *Belief-Desire-Intention* (BDI) model of agency is an architecture based on Bratman's theory of practical reasoning. Hierarchical Task Network (HTN) decomposition on the other hand is a *planning* technique which has its roots in classical planning systems such as STRIPS. Despite being used for different purposes, HTN and BDI systems appear to have a lot of similarities in the problem solving approaches they adopt. This paper presents these similarities. A systematic method for mapping between the two systems is developed, and experimental results for different kinds of environments are presented.

1 Introduction

The *Belief-Desire-Intention* (BDI) [1] agent development framework (e.g. JACK [2] and PRS [3]) appears in many ways to be very similar to the Hierarchical Task Network (HTN) approach to planning (e.g. UMCP [4], SHOP [5]), although the former arises out of the multi-agent systems community, while the latter arises out of the planning community.

Both BDI and HTN systems use a notion of decomposition, and flexible composition of parts, although BDI systems are primarily used for deciding goal directed agent actions in dynamic environments, while HTN systems are used for formulating a plan which is later executed.

Earlier research (e.g. [6, 7]) mentions similarities between HTN planning and BDI style execution. The work most closely related to ours is in the *ACT* formalisms of the Cypress system [7]. Work done for Cypress is different to our work in that ACT is an *interlingua* that enables the two systems to share information, whereas we provide a mapping between HTN and BDI systems. Furthermore, the HTN planner in Cypress is a partial-order HTN planner, whereas we use a total-order (hereafter referred to simply as HTN) HTN planner.

Despite the close similarities of HTN and BDI systems, there does not appear to be any work done which systematically contrasts and compares the core approaches and algorithms developed in the two communities. This paper provides a detailed comparison between the two approaches, including a mapping from HTN to BDI representations. We also explore the efficiency of the underlying algorithms of the two kinds of systems, via experimentation in varying situations. This work provides a basis on which application developers can choose the preferred implementation platform, as well as providing some insights into how frameworks in either paradigm may be improved.

2 Similarities and Differences between HTN and BDI

Both HTN planners and BDI agent execution systems create solutions by decomposing high level tasks (or goals) into more specific tasks and primitive actions. The tasks as well as the decomposition methods (or plans) are specified by the programmer in both cases.

However, the systems (usually) serve a different purpose in that HTN planners are used to efficiently find a plan, which can then be executed, whereas BDI systems are used to guide execution in real time. There is some work on interleaving planning and execution, using HTN planners [6], which is then very similar in style to BDI execution and is therefore suitable for guiding actions in dynamic environments. BDI systems can also be used to search for a solution before executing it, in situations where this is appropriate or desirable.
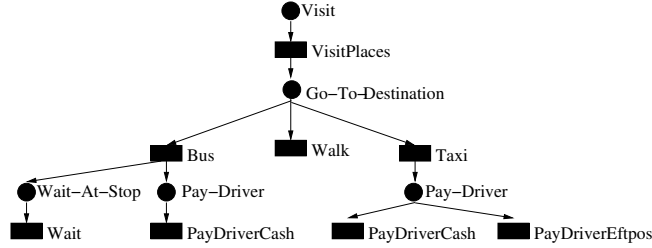

Fig. 1. Goal-plan hierarchy in BDI or Task-network in HTN.

An example of a goal-plan hierarchy in BDI or a task network in HTN is shown in Figure 1[1]. In this Figure, circles represent BDI goals or HTN abstract tasks and rectangles represent BDI plans or HTN methods. The hierarchy begins by having a goal/task to make a visit which can be achieved by (decomposed into) the *VisitPlaces* plan (method). This plan (method) has a goal (task) to go to the destination which in turn can be achieved by (decomposed using) one of the three plans (methods): *Bus*, *Walk* or *Taxi*, etc.

The fact that this structure can equally well represent an HTN task network, or a BDI goal-plan tree, indicates a certain similarity between the systems. Also, the approach of returning to try an alternative path through the tree if difficulties are encountered, is similar in both cases.

However reasons for "backtracking" in this structure are subtly different. BDI systems will backtrack only if there has been some failure - usually caused by some change in the environment, or by the lack of complete predictability of actions. HTN systems backtrack when a solution that has been pursued, turns out not to work. There is no opportunity for discovering problems within the environment, during the planning process.

If we are to compare execution of HTN and BDI systems we need to choose a particular HTN and BDI system to work with, and then map programs between the two systems. The HTN system we use is JSHOP which is a Java version of the SHOP planner. JSHOP is being used by the *Naval Research Laboratory for Noncombatant*

---

[1] This example was taken from [5] and extended.

Evacuation Operations [2]. SHOP2 is a generalization of SHOP/JSHOP that won one of the top four prizes in the 2002 International Planning Competition.

We have developed a systematic translation that we have used to convert JSHOP programs to JACK programs. The translation deals with the main entities of JSHOP, which are methods, operators and axioms [5], whereas the main entities of BDI according to [8], are plans, goals or events and beliefs[3].
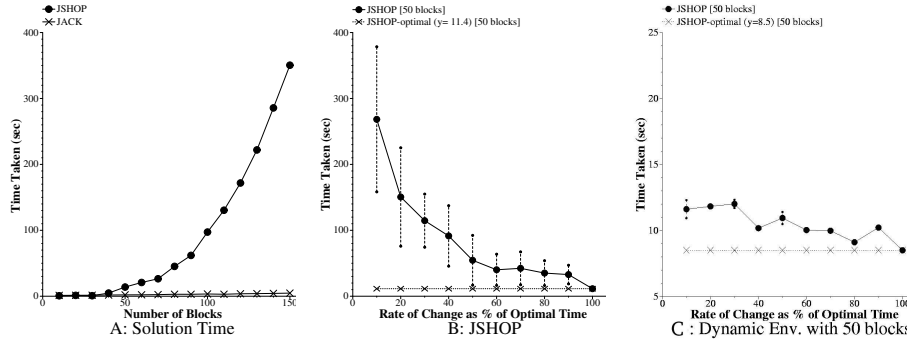
3   Experimental Comparison

Fig. 2. A: Solution time for JACK and JSHOP with increasing number of blocks, B: and C: JSHOP and modified JSHOP (respectively) in a dynamic environment

In its original form, BDI systems were designed for use in highly dynamic environments, and HTN systems were designed for use when guaranteed solutions were necessary. Some research also focussed on building hybrid systems that combine the useful (e.g. [6, 7]) properties of each system. In this section, we provide emperical foundations for past and future work, by analysing how each system performs in different environments.

In order to compare the performance of BDI and HTN algorithms under differing problem sizes and environmental situations, we took examples of blocks world encoding provided with JSHOP, extended these, and mapped to JACK, using the mapping mentioned previously. We then ran experiments to explore time and memory usage in static and dynamic environments. The Blocks World domain was used because it can easily be scaled to a range of problem sizes, and also because tested JSHOP encodings [5] for the problem were already provided.

The JSHOP blocks-world domain representation as well as sample problems from 10 blocks to 100 blocks was provided with the JSHOP planner (originally obtained from [9]). We used the problems provided and created additional problems for 110, 120, 130, 140 and 150 blocks by combining the 100 blocks problem with 10 blocks problems (including block renumbering). We randomly selected one problem of each

---

[2] http://www.cs.umd.edu/projects/shop/description.html

[3] We leave out the details due to space restrictions. See http://www.cs.rmit.edu.au/ ldesilva for a more detailed paper.

size, for problems of size 10-100. Each problem was specified in terms of the start position of all blocks, and a goal state specifying the position of all blocks.

The program encoding consisted of one compound task *move*, with four different decompositions for achieving it, each having a different precondition. The primitive actions consisted of four operators; *pickup*, *putdown*, *stack* and *unstack*. Due to space constraints, refer to [9] for full details. An axiom was used to indicate whether a block needed to be moved. This *need-to-move(x)* axiom (where x is a block) evaluates to true or false based on whether one of a number of conditions are met. For example, *need-to-move(x)* would be true if x is on the table and there is a goal state requiring x to be on some other block. This heuristic allowed problems in this particular encoding of the blocks-world to be solved without needing HTN style lookahead, since standard BDI reasoning is not capable of such lookahead.

The mapping techniques we had developed were then used to translate from JSHOP to JACK representation for each problem.

The experiments were run on a dedicated machine, running Linux Red Hat 8.0, with an Intel Pentium IV - 2GHz CPU, and 512MB of memory. Each experiment was an average of 10 runs. Measurements taken were time[4]/memory required to create a solution. In JACK, the solution is found through execution in a simulator, whereas JSHOP produces the solution as a list of actions, which is then executed in the simulator.

The experiments performed explored: 1) *Runtime in static environments of size 10-150 blocks*, 2) *Runtime in dynamic environments of size 30 - 50 blocks*, 3) *Memory usage in environments of size 10-100 blocks*.

3.1 Runtime in static environment
The first experiment compared the time taken in both systems to find one solution, with an increasing number of blocks. Figure 2A shows these results.

For Figure 2A, DeltaGraph[5] showed that the time taken by JSHOP is approximately $0.03x^2$, which is quadratic, while time taken by JACK is approximately $0.02x$, which is linear. Statistical results also confirmed that these two graphs were significantly different.

Further experiments to understand JSHOP's quadratic performance revealed that JSHOP's precondition evaluation algorithm took at least 75 percent of the processing time, in particular, its *unification* algorithm used from within *theorem prover*. The unification algorithm was not complex in itself, but had a high frequency of calls. A more complete analysis of runtime in a static environment is left as future work.

Experiments for the memory usage of JSHOP and JACK using problem sizes of 10-100 blocks showed the same pattern as that of Figure 2A for unmodified JSHOP.

3.2 Runtime in dynamic environment
For these experiments a dynamic Blocks World environment was used, where a random move action was introduced periodically. This simulated a situation where the environment is changing outside of the control or actions of the agent. The externally introduced move action was selected by randomly choosing (source and destination) from among the blocks that were currently clear. Differing rates of change were used, with

---

[4] Using the *time* command, the CPU + system times spent in execution.

[5] http://www.redrocksw.com/deltagraph/

the slowest rate being the time taken to execute the entire solution in a static environment. We call this time the *optimal time* (refer to Figure 2A). Slower change than this would of course have no effect. The dynamism was increased each time by 10 percent of the slowest rate.

For these experiments, executing the solution found by JSHOP was significant in order to determine whether it actually reached the goal, given the changing environment[6]. Failure could occur either when a planned action failed (for example due to a block to be picked up, no longer being clear), or when the whole plan had been executed, but on checking the goal had actually not been accomplished, due to environmental changes. At the point of failure, JSHOP replanned from the new environmental state.

Figure 2B shows the time taken by JSHOP to find a solution for a problem of size 50 blocks, as the dynamism in the environment decreases.[7] The horizontal dotted line crossing the y axis at y=11.4, in Figure 2B, is the optimal time. As the dynamism increases, the time taken to find a solution also increases at a rate of approximately x^3.

This is because every time the environment changes, JSHOP has to replan for the new environmental state, although usually it would have moved somewhat closer to the goal. Therefore as the dynamism increases, the number of plans generated in order to find a solution is likely to increase. The large standard deviation (dashed vertical lines) as the dynamism increases is due to the variability in how much of an effect the environmental change has on plans being created, due to whereabouts in a plan, a failure occurs.

Experiments with JACK in the same dynamic environment was linear, which showed that the behaviour of JACK does not appear to be significantly affected by the rate at which the environment changes (figure not shown due to space constraints). This is to be expected as plans are chosen in the current environment immediately prior to execution. In addition a plan choice commits only to relatively few steps, and so if it is affected by environmental change, only relatively little time is lost. Experiments also shows that there is not much standard deviation in the results, and that the standard deviation is consistent, even with an increasing rate of change.

There is an increasing amount of work in adapting HTN planners to interleave execution with planning (e.g. [6]), making them operate in some ways more like BDI agent systems. We adapted JSHOP to execute each method directly after decomposition, obtaining the experimental results shown in Figure 2C. Note that y=8.5 seconds was the optimal time for finding a solution, when the first decomposition (with at least one action) was immediately executed (as opposed to forming a complete solution).

We found the degradation of the system as dynamism increases to be similar to that of JACK. Further, statistical tests showed that the behaviour of modified JSHOP is significantly different to the behaviour of the original version of Figure 2B.

4 Discussion and Future Work

On the examples tested, the growth rate of JACK of finding a solution, compared to JSHOP as problem size increases, is linear as opposed to polynomial. This has

---

[6] The changing environment here means the external environment that JSHOP finds solutions for and not changes to the initial state during planning.

[7] Results were similar for 30 and 40 blocks.

a significant impact for large applications. Future work could include an analysis of JACK's complexity, in particular, its context condition evaluation, for a comparison with JSHOP's complexity in [4]. A complexity analysis may also enable HTN systems to benefit from faster algorithms used by BDI systems (or at least by JACK).

Although our comparison used a single implementation of a BDI and total-order HTN each system, we emphasise that we considered the formalisms [8, 5] of two state of the art systems for our mapping.

Due to the similarity of the core mechanisms in the two paradigms, each can borrow some strengths from the other. Since BDI systems allow real time behaviour in quite dynamic domains, HTN systems can be made to behave like BDI systems in dynamic domains by executing methods immediately after decomposition. Alternatively, BDI agents could use HTN planning in environments when lookahead analysis is necessary to provide guaranteed solutions. In situations where the environment is not highly dynamic, BDI agents could use HTN lookahead to anticipate and avoid branches in the BDI hierarchy that would prevent the agent from achieving a goal.

We also acknowledge that both types of systems have strengths and functionality not covered in this work, which may well make them the system of choice for a particular application.

5 Acknowledgements

References

1. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, San Francisco (1995)
2. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack Intelligent Agents - components for intelligent agents in java. AgentLink News Letter, Agent Oriented Software Pty. Ltd, melbourne (1999)
3. Georgeff, M., Ingrand, F.: Decision making in an embedded reasoning system. In: Proceedings of the International Joint Conference on Aritificial Intelligence. (1989) 972–978
4. Erol, K., Hendler, J.A., Nau, D.S.: Complexity results for HTN planning. Annals of Mathematics and Artificial Intelligence **18** (1996) 69–93
5. Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H.: SHOP: Simple Hierarchical Ordered Planner. In: Proceedings of the International Joint Conference on AI. (1999) 968–973
6. Paolucci, M., Shehory, O., Sycara, K.P., Kalp, D., Pannu, A.: A planning component for RETSINA agents. In: Agent Theories, Architectures, and Languages. (1999) 147–161
7. Wilkins, D.E., Myers, K.L., Lowrance, J.D., Wesley, L.P.: Planning and reacting in uncertain and dynamic environments. Journal of Experimental and Theoretical AI **7** (1995) 197–227
8. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), Toulouse, France. (2002)
9. Gupta, N., Nau, D.S.: On the complexity of blocks-world planning. Artificial Intelligence **56** (1992) 223–254