Planner9, a HTN planner distributed on groups of miniature mobile robots

Stéphane Magnenat, Martin Voelkle, and Francesco Mondada

LSRO laboratory, EPFL, 1015 Lausanne, Switzerland,
stephane at magnenat dot net,
http://mobots.epfl.ch

Abstract. Autonomous mobile robots are promising tools for operations in environments that are difficult to access for humans. When these environments are dynamic and non-deterministic, like in collapsed buildings, the robots must coordinate their actions and the use of resources using planning. This paper presents Planner9, a hierarchical task network (HTN) planner that runs on groups of miniature mobile robots. These robots have limited computational power and memory, but are well connected through Wi-Fi. Planner9 takes advantage of this connectivity to distribute the planning over different robots. We have adapted the HTN algorithm to perform parallel search using A* and to limit the number of search nodes through lifting. We show that Planner9 scales well with the number of robots, even on non-linear tasks that involve recursions in their decompositions. We show that contrary to JSHOP2, Planner9 finds optimal plans.

1 Introduction

Autonomous mobile robots are promising tools for operations in environments that are difficult to access for humans. In particular, groups of miniature low cost robots are well suited for missions in hazardous environments; for instance to do search and rescue in partially destroyed structures. Indeed, small robots can pass through small holes to access most locations. Several robots can collaborate to free a path, for instance by moving objects or by extinguishing fires. Multiple physical robots provide redundancy against contingencies resulting from the instability of the environment. As the robots are inexpensive, the loss of some of them is acceptable. In this context of rescue operations, most of previous works focused on reactive controllers, see for instance [1,2]. However, real-world robotics tasks that involve manipulation of the environment cannot be performed by reactive controllers only, because the actions of the robots might modify the state of the world irreversibly into situations that prevent the completion of the mission. For instance in a rescue scenario, robots must never irreversibly fill the only passageway to humans trapped in a room, regardless of how this action might help completing other tasks. To alter the world the right way, the robots must reason about their course of action; the common way to do so is to use an automated planner.

This paper presents Planner9, a distributed hierarchical task network (HTN) planner [3, ch. 11] that runs on groups of miniature mobile robots. These robots are good models of expendable field robots because they are affordable and all-terrain [4]. They are built around a smartphone-level computer which provides one tenth of the power for one twentieth of the energy consumption of a laptop computer. This computer runs Linux, and provides a Wi-Fi connection to its peers. Planner9 takes advantage of this large bandwidth on CPU ratio to distribute the planning over different robots.

In the results section, we show that Planner9 scales well with the number of robots. We study how the planning durations vary when we change the complexity of the environment. We compare Planner9 to JSHOP2, a free and an often referenced HTN planner.

2 Related Work

A vast literature exists on planning with multiple agents. However, the choice of a particular algorithm is a trade-off between several factors such as planning expressiveness, distributivity, bandwidth consumption, and speed of execution [5]. In this section we focus on the approaches that are implementable on real robots with limited resources.

To distribute planning among different agents, one can run an independent planner on each agent. Works in simulated robotics soccer have used HTN planning that way [6]. However, in general this solution cannot improve the time nor reduce the memory required to solve a specific problem. In the direction of distributing the planning process, [7] has integrated the SHOP HTN planner within a distributed agent framework. The resulting A-SHOP planner uses the provided infrastructure to query the state of the world, evaluate preconditions, apply effects, and estimate potential states from remote agents. However, in this work the planning itself is still centralized. Theoretical works in multi-agent systems have shown that it is possible to integrate the distributed aspect in the planning algorithm [8,9]. For instance [10] has proposed a market-based approach where *and/or* trees of tasks are exchanged between agents. These authors acknowledge the interest of more expressive planning but address the issues of distributivity and scalability first ("Further research will also investigate further generalization of the tree structures and task constraints." [11, p. 25]). Recent works in HTN have proposed stratified planning where remote agents plan subtasks and report them to a master. In [12], finding the final plan is the result of the exchange of proposals and counter proposals between agents. In [13], the child agents are also responsible for execution, and interleave planning, execution, and replanning. These methods require a large number of synchronization messages. On the contrary, Planner9 considers the different robots as a computer cluster and distributes the planning of any task to any robot and thus takes advantage of all the available computational power using simple synchronization.

3 Materials and Methods

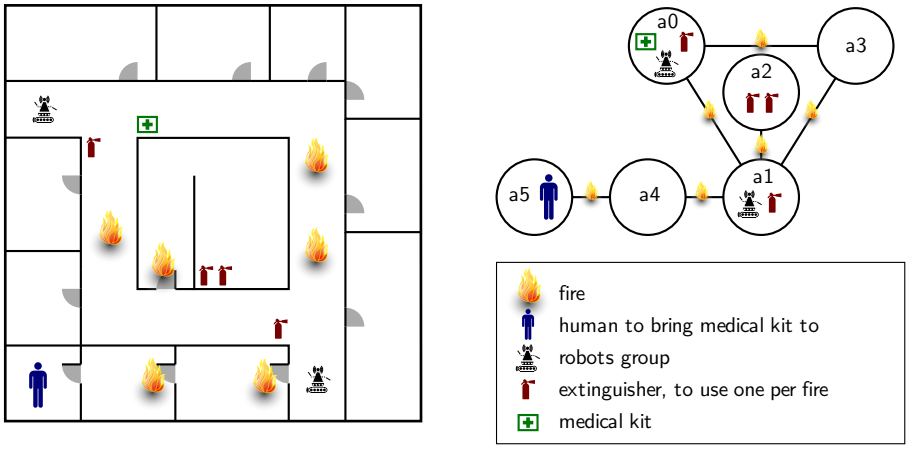

Fig. 1. Search and rescue scenario (left) with abstract representation (right). The robots are dropped into a damaged building. They must bring a medical kit to the humans trapped in the lower left room. To do so, they must extinguish the fires in the right order; otherwise they would fail the rescue operation as there are not enough extinguishers readily available to put out all fires.

To measure the performances of Planner9, we have developed a small search and rescue scenario (Fig. 1). In this scenario, groups of robots are dropped into a damaged building and must bring a medical kit to humans trapped in a room. Multiple fires block the ways; robots can use fire extinguishers to put them out, but an extinguisher can put out only one fire. It is a typical situation where reactive controllers would fail because there are not enough extinguishers directly available to put out all fires. The robots must use the right extinguishers on the right fire in the right order and thus need a plan (Fig. 2). We believe that this scenario is representative of the complex tasks that would require a planner. Here we suppose that the robots know the locations of fires, extinguishers, objects, and people. To discover them in a real-world situation, we can imagine to deploy exploration robots that would map the environment.

We conceived Planner9 to run on our latest generation of miniature mobile robots, which comes down from the S-bot robot [4]. Only one prototype is currently available, yet dozens of robots are in production. We have compiled, run, and benchmarked Planner9 on this prototype. To simulate more robots, we then run multiple slave processes of the planning algorithm on a multi-core computer while another computer runs a master control program. We use cpulimit[1] to only

[1] `http://cpulimit.sourceforge.net/`

```
take(r0, e0)                          a0, a1, a2, a3, a4, a5 : rooms
extinguish(a1, a0, r0, e0)
take(r1, e1)                          r0, r1 : groups of robots initially
extinguish(a2, a1, r1, e1)            in a0 and a1 respectively
move(a2, r0)
take(r0, e2a)                         e0, e1, e2a, e2b : extinguishers
move(a1, r0)                          initially in a0, a1, a2, and a2
extinguish(a4, a1, r0, e2a)           respectively
move(a2, r0)
take(r0, e2b)                         o0 : medical kit, initially in a0
move(a4, r0)
extinguish(a5, a4, r0, e2b)
move(a0, r1)
take(r1, o0)
move(a5, r1)
drop(r1, o0)
```

Fig. 2. Solution plan for the scenario presented in Fig. 1. `take` lets a group of robots pick up an object. `extinguish` puts out a fire between two rooms. `move` displaces a robot group to a room. `drop` puts down the object the robots hold.

allow as much processing power as available on the robot, which corresponds to 5 % of an Intel Core2 at 2.83 GHz. Thanks to ulimit[2], we limit the memory to 100 MB, which is the amount available on the robot. Finally, we divide the available Wi-Fi bandwidth (19 Mbps using IEEE 802.11g) by the number of slaves and limit the network bandwidth using trickle[3]. This way, our measurements are representative of the expected performances on the robots. This approach works as long as all slaves plan for a long duration. When we increase the number of slaves, the time to find a solution decreases which is a problem. Indeed, cpulimit and trickle impose limitations by pausing and restarting execution and data transmission. They interfere with our measurements when slaves find solutions in the same order of magnitude of time as these tools operate. We have observed such interferences starting from 8 slaves. Therefore, we limit our scalability measurements to 7 slaves.

We perform two experiments. First we measure the scalability of Planner9 by varying the number of planning slaves (simulated number of robots), as explained in the previous section. We perform 100 runs for each point, and show the average and the standard deviation. Second, we vary the difficulty of the planning by adding elements to the world that are useless to the robots, as show in Fig. 3. We first add two empty rooms, then we add extinguishers in these rooms and a medical kit in a3. These elements add more branches to the search tree without affecting the solution. For each case, we perform 100 runs using 1 and 7 slaves. We

[2] `http://www.linuxhowtos.org/TipsandTricks/ulimit.htm`
[3] `http://monkey.org/~marius/pages/?page=trickle`

also solve the same problem using JSHOP2[4] [14], a free and an often referenced HTN planner implemented in Java. As JSHOP2 is a depth-first search planner, we add bookkeeping actions that prevent infinite recursions. We do not count these actions when comparing plans sizes. We substract the startup time of JSHOP2 (time to decompose an empty task) to put it on an equal basis with Planner9.

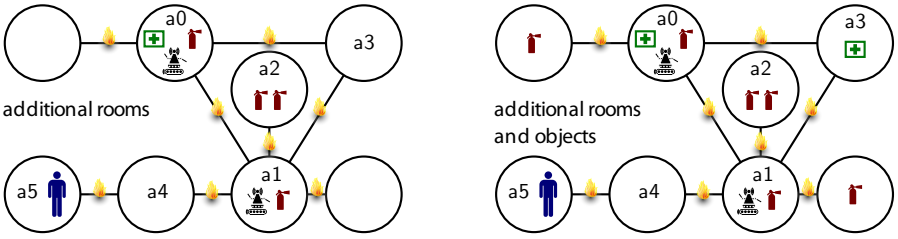

Fig. 3. Scenarios with additional elements useless to the robots. These elements add more branches to the search tree without affecting the solution.

4 Planner9

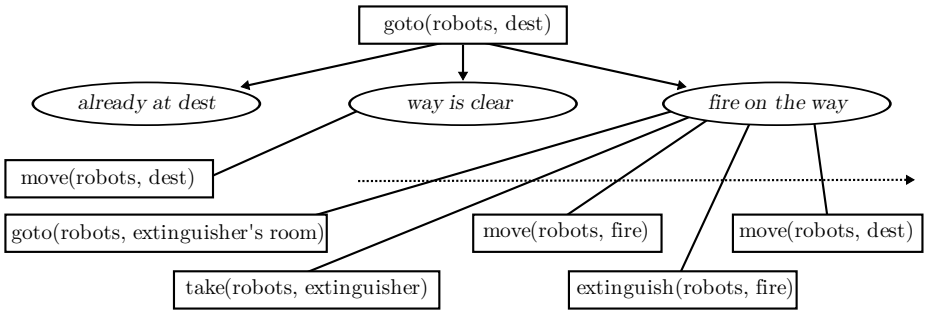

Fig. 4. Methods for going to a room by recursively putting out fires on the way.

Planner9 is a HTN planner. A HTN planner decomposes a goal task into subtasks until it finds a sequence of actions that the robots can perform. The planner knows the available methods and their possible decompositions, like in Fig. 4. When given the goal task and the initial state of the world, the HTN planner seeks an admissible sequence of actions. The actions can affect the state of the world; so the planner records these alterations.

[4] `http://sourceforge.net/projects/shop`

Planner9 plans partially-ordered graphs of tasks using forward decomposition, as in [3, p. 243]. It keeps track of each possible decomposition in a different search node. Planner9 starts planning with a single node containing the goal task and the initial state of the world. When visiting a node, Planner9 iterates through all tasks that have no predecessor. If the task is an action, it applies this action to the current state of the world and stores the action as part of the plan. Otherwise, Planner9 instantiates the different possible decompositions of the task. This process goes on until there is no more node left or until Planner9 has found a node with no more task to decompose.

The state of the world consists of a set of n-ary relations over a set of values. The application of an action affects these relations. The values represent things from the real world, like rooms or robots in Fig. 4. In the latter, the `move` action will update the `isIn` relation between the robots and the rooms. The planner creates variables when decomposing tasks. For example in Fig. 4, the `goto` task can be decomposed using the *fire on the way* alternative. This decomposition introduces new variables: the extinguisher to use and the room where the extinguisher is located. The decomposition has preconditions over these variables that the state of the world must satisfy. For instance, the extinguisher must be located in an accessible room. When Planner9 decomposes a task, it performs *lifting*: it accumulates its preconditions for delayed check. Planner9 assigns a value to a variable only when an action changes a relation this variable appears in. For every variable assignations allowed by the accumulated preconditions, Planner9 updates the state and creates a new search node. To do so in an efficient way, it assigns values using DPLL [15]. The use of lifting is one of the improvements of Planner9 over the basic HTN algorithm. It results in fewer search nodes and more processing per node, which is desirable for parallelization.

Planner9 chooses the node to visit by selecting the least expensive one using A* [16]. As cost it adds the total cost of the decomposition so far (path-cost in A*) and the number of remaining tasks to be decomposed (heuristic-cost in A*). One advantage of A* with respect to a depth-first search is to allow free recursions in the definition of the planning domain. This is useful in robotics because real-world problems are often expressed in a recursive way, like in Fig. 4. Moreover, as each node is independent, Planner9 can distribute the A* search over the network using a master/slave architecture.

The slaves run the algorithm described in the previous paragraphs and report periodically the cost of their cheapest nodes to the master. When the cost differences between slaves are significant, the master requests nodes from the slave with the lowest cost. It then transfers them to the slave with the highest cost. As our results show, this load-balancing algorithm is simple yet efficient. Slave robots can appear on and disappear from the network dynamically, for instance when they boot or because their batteries are discharged. The master discovers available slaves dynamically using the Zeroconf protocol[5], which is based on broadcast announcing. Thanks to it, Planner9 does not require any central registry and can use any robot available on the local network.

[5] `http://www.zeroconf.org/`

Planner9 is implemented in C++, is open source[6], and runs on embedded Linux. It only depends on the C++ standard library, the boost library[7], and Qt Embedded[8], which are all open source. Planner9 is thus easy to embed in any robot running a modern Linux board.

5 Results

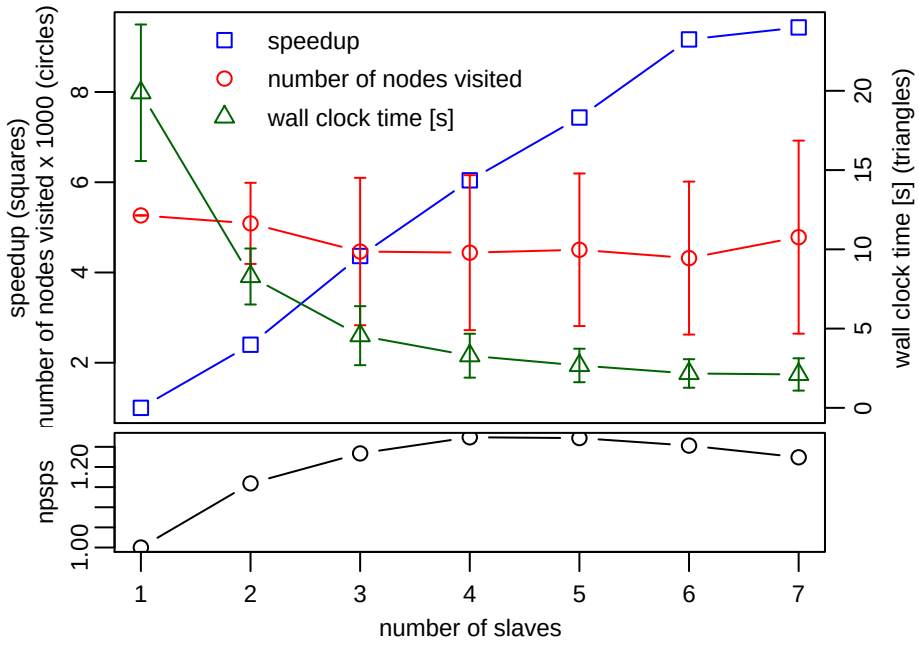

Fig. 5. The scalability of Planner9 with respect to the number of slaves. The scenario is shown in Fig. 1. The error bars show the standard deviation over 100 runs. In the second plot, *npsps* represents the number of nodes visited per second per slave, renormalized.

Fig. 5 shows the measures of the scalability of Planner9. The first plot shows that the performances scale nicely with the number of slaves. The speedup, which is the planning time using n slaves divided by the time using 1 slave, is even super linear. We analyse this using the second plot, which shows the number of

[6] One can access the source tree using git; to retrieve it, type: `git clone git://gitorious.org/planner9/planner9.git`. We performed the experiments using revision `a501cfd704e4c759a0d83ea27dfcc85be53929ef`.

[7] `http://www.boost.org`

[8] `http://www.qtsoftware.com/products/`

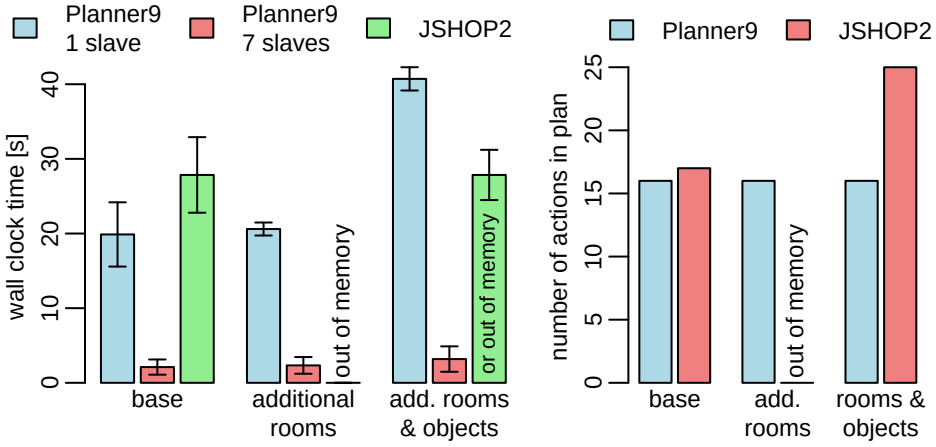

Fig. 6. The scalability of Planner9 with respect to the complexity of the environment. The scenario are shown in Fig. 3. The error bars show the standard deviation over 100 runs. JSHOP2 fails to find a plan when we add rooms, but sometimes succeeds when we further add objects, as we explain in the main text.

nodes visited per second per slave, renormalized. When we increase the number of slaves, each can processes more nodes per second. The cause of the super linearity is a combination of the structure of the problem [17] and a memory cache effect. Indeed, by distributing the planning, each slave holds fewer search nodes in memory. Saving memory reduces the strain on the CPU cache and on the memory allocator, which results in a better performance. When the number of slaves grows further, the performance deteriorates. We attribute this effect to the load balancing, which increases the average time to find a solution because it moves low-cost nodes around.

Fig. 6 shows the measures of the ability of Planner9 and JSHOP2 to cope with environments of increasing difficulty. On the basic environment, Planner9 with 1 slave is faster than JSHOP2; using 7 slaves, it is one order of magnitude faster. Moreover, Planner9 uses both groups of robots while JSHOP2 extinguishes all fires with `r0`, which adds one more move action to the plan. Adding two empty rooms does not disturb Planner9 much, but JSHOP2 cannot cope: it exhausts its memory before finding any solution, even if we give it 2 GB instead of the 100 MB available on the robot. When we add useless objects in the empty rooms, the planning time of Planner9 increases but does not explode. In this environment, JSHOP2 either finds a solution quickly or exhausts its memory, depending on the order of appearance of the new elements in its initial state of the world. If we declare the new elements (rooms and objects) before the basic environment, JSHOP2 finds a solution. If we declare them after, it fails. Moreover, when it finds a solution it is a long plan with a lot of useless actions, such as using one extinguisher to access a room to get another one.

These results show that Planner9 scales well with the complexity of the environment, compared to JSHOP2. We attribute these excellent performances to the lifting which keeps different values assignments in a single node by abstracting them using a variable. Planner9 assigns a value later, when it has collected more constraints on this variable. On the contrary, JSHOP2 assigns values early and performs depth-first search. If by chance the first search branches lead to the solution, JSHOP2 finds it quickly. But otherwise, it must explore an exponentially large number of branches before finding a solution.

6 Future Work

The next step is of course to run Planner9 completely on our robots and confirm its scalability. Then, we will integrate Planner9 with the perception subsystem to create the initial state of the world and implement the execution of plans using the robots' actuators. We want to study the expectations and the limitations of using Planner9 to bring reasoning to collective robotics.

Planner9 is currently a basic HTN planner and does not provide any extension such as probabilistic planning or conditional actions. The rationale is that planning real-world scenarios is computationally intensive and we want it to run live on the robots themselves. Moreover, we can make good use of the availability of multiple robots to reduce the uncertainty of the world perception through concurrent sensing from different locations. Merging these would produce a robust estimation of the state of the world for planning at symbolic level. Nevertheless, the parallelization capabilities of Planner9 would work with extensions as well so we can implement them should the need arise.

7 Conclusion

We have presented Planner9, a HTN planner that dynamically distributes its processing to multiple mobile robots by considering them as a computer cluster. To do so we have enhanced the HTN algorithm with an A* search and with the lifting of the tasks' preconditions. Previous works have explored how to distribute the decomposition of a particular subtask, but Planner9 distributes the processing of any task and thus benefits from all available computational power. Albeit simple, this mechanism is efficient as our results show. When compared to JSHOP2, a milestone in HTN research, Planner9 always finds optimals plans; while JSHOP2 only sometimes finds plans, and they are sub-optimal. Because Planner9 runs on inexpensive robots, we believe that it is a firm step towards bringing intelligence to autonomous mobile robots in non-industrial environments.

8 Acknowledgements

We thank Valentin Longchamp for compiling Planner9 on the prototype of the robot. We thank the Open Clip Art library[9] and Valessio Soares de Brito for the

[9] `http://www.openclipart.org/`

graphics elements used in Fig. 1 and Fig. 3. We thank Emmanuel Eckard and Basilio Noris for proofreading the manuscript.

References

1. Kumar, V., Rus, D., Singh, S.: Robot and sensor networks for first responders. Pervasive Computing, IEEE **3**(4) (Oct.-Dec. 2004) 24–33
2. Stormont, D.: Autonomous rescue robot swarms for first responders. In: Computational Intelligence for Homeland Security and Personal Safety. Proceedings of the 2005 IEEE International Conference on, IEEE Press (2005) 151–157
3. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann (2004)
4. Mondada, F., Pettinaro, G.C., Guignard, A., Kwee, I., Floreano, D., Deneubourg, J.L., Nolfi, S., Gambardella, L., Dorigo, M.: SWARM-BOT: a New Distributed Robotic Concept. Autonomous Robots, special Issue on Swarm Robotics **17**(2-3) (2004) 193–221
5. Erol, K., Hendler, J.A., Nau, D.S.: HTN Planning: Complexity and Expressivity. In: AAAI, AAAI Press (1994) 1123–1128
6. Obst, O., Boedecker, J.: Flexible Coordination of Multiagent Team Behavior Using HTN Planning. Lecture Notes in Computer Science **4020** (2006) 521–528
7. Dix, J., Munoz-Avila, H., Nau, D., Zhang, L.: IMPACTing SHOP: Putting an AI planner into a multi-agent environment. Annals of Mathematics and Artificial Intelligence **37**(4) (2003) 381–407
8. Durfee, E.: Distributed problem solving and planning. Multiagent systems: a modern approach to distributed artificial intelligence (1999) 121–164
9. DesJardins, M., Durfee, E., Ortiz, C., Wolverton, M.: A survey of research in distributed, continual planning. AI Magazine (1999)
10. Dias, M., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE **94**(7) (July 2006) 1257–1270
11. Zlot, R., Stentz, A.: Market-based multirobot coordination for complex tasks. The International Journal of Robotics Research **25**(1) (2006) 73–102
12. Pellier, D., Fiorino, H.: A Unified Framework Based on HTN and POP Approaches for Multi-Agent Planning. In: Intelligent Agent Technology. IEEE/WIC/ACM International Conference on, IEEE Press (Nov. 2007) 285–288
13. Hayashi, H., Tokura, S., Ozaki, F.: Towards Real-World HTN Planning Agents. Knowledge Processing and Decision Making in Agent-based Systems **170** (2009) 13–41
14. Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research **20**(1) (2003) 379–404
15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7) (1962) 394–397
16. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on **4** 100–107
17. Rao, V.N., Kumar, V.: Superlinear speedup in parallel state-space search. In: Proceedings of the Eighth Conference on Foundations of Software Technology and Theoretical Computer Science, London, UK, Springer-Verlag (1988) 161–174