Flexible Behavior for Worker Units in Real-time Strategy Games Using STRIPS Planning

Ioannis Vlachopoulos[1], Stavros Vassos[2], and Manolis Koubarakis[1]

[1] Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece, {johnvl,koubarak}@di.uoa.gr
[2] Department of Computer, Control, and Management Engineering Sapienza University of Rome, Italy, vassos@dis.uniroma1.it

Abstract. In this paper we investigate how STRIPS planning techniques can be used to enhance the behavior of worker units that are common in real-time strategy (RTS) video games. Worker units are typically instructed to carry out simple tasks such as moving to destinations or mining for a type of resource. In this work we investigate how this interaction can be extended by providing the human player with the capability of instructing the worker unit to achieve simple goals. We introduce the "Smart Workers" STRIPS planning domain, and generate a series of planning problems of increasing difficulty and size. We use these problem sets to evaluate the conditions under which this idea can be used in practice in a real video game. The evaluation is performed using a STRIPS planner that is implemented inside a commercial video game development framework.

1 Introduction

In real-time strategy RTS video games the human player is, more or less, in charge of a kingdom over a large game map where other kingdoms also co-exist and fight over resources and domination. This amounts to controlling a variety of buildings, units, and resources that are located in the map, with the aim to expand their territory and eliminate the other kingdoms in the map.

Worker units are a special type of units that are typically instructed to carry out simple tasks such as moving to destinations and mining for a type of resource. Such tasks are completely specified by the human player using direct commands that instruct the worker unit to perform each of the atomic actions separately. In this work we investigate how this can be extended by providing the human player with the capability of instructing the worker unit to achieve simple goals. The idea is to employ worker units with an appropriate STRIPS representation [8] of the game-world so that these commands-goals can be resolved by taking advantage of methods for real-time planning.

This practical application domain seems well-suited for experimenting with real-time planning techniques as i) it features a variety of objects that the worker units may need to interact with, including different types of buildings, resources, as well as other units, ii) it requires that the worker unit performs a different sequence of actions for achieving the same objective in the variety of situations that may occur in the game-world, and iii) the gameplay is such that an instantaneous reactive response is not a strong requirement; some upgraded worker units with deliberation capabilities may take

even a couple of seconds to "think" before executing the command given by the human player without this breaking the believability of the characters.

In order to explore this idea we developed the "Smart Workers RTS mini-game". The game-world of this conceptual game is slightly more refined than usual RTS video games, in that some actions require the worker unit to hold a specific tool so that to be able to perform the action. Different buildings provide different tools and resources, and, similar to usual RTS games, depending on which buildings the human player decides to build consuming ones resources, different tools are provided for worker units to use. For instance, food rations (which are necessary for feeding the units of the kingdom) can be produced in various ways: by hunting for animals and then cooking the raw-meat, by harvesting a rice farm, etc. For each of these actions the worker unit may need to visit a building first to acquire the specific tool needed, e.g., getting some weapon from the armory before going to hunt for animals.

We investigated how flexible behavior for worker units can be handled via planning using heuristic state-space search. A usual concern among game developers against using sophisticated approaches for decision making is that it is often faster to customize the domain specification so that a simpler approach can be applied. In order to explore this, we experimented with the idea of ordering the action types with which applicable actions are generated so that to help simple search methods find the goal. Our results show that even in this relatively simple domain this practical approach is very unstable in its performance: there are scenarios where it actually performs better than a planning approach but also others where it performs very poorly. On the other hand, it becomes obvious that for domains like this where there is a significant level of variety in the game-world specification, the sophisticated academic AI methods prove to be more robust and practical.

2 STRIPS planning

In a classical planning we are given i) an initial state, ii) a goal condition, and iii) a set of available actions in terms of preconditions and effects, and we want to find a sequence of actions such that if we execute each action in sequence starting from the initial state, we will result in a state that satisfies the goal condition. In this work we will focus on the most basic type of planning problems and the STRIPS formalism which allows us to model a planning problem using first-order logic literals in a practical way [8].

In STRIPS, a state is represented as a set of ground first-order logic literals following a closed world assumption (CWA). That is, no variables or function symbols can be mentioned in a literal, only constants. Also, by the CWA principle, the presence of a ground literal in the set implies that the literal holds in the state while its negative version does not hold in the state, and absence of a ground literal in the set means that the literal does not hold in the state while its negative version holds in the state. In the Smart Workers domain that we describe in Section 3, the following literals may be used to represent that (from the perspective of a particular worker unit), the worker unit is holding a sword, there is a wild boar in the forest, and the fact that raw meat can be

converted into food:

$$\textit{holding}(sword_2),\ \textit{lives-in}(boar_2, forest_1)$$
$$\textit{is-converted-to}(raw\text{-}meat, food\text{-}ration),$$

The available actions are represented by action schemas, each of which characterize a set of possible ground actions using variables as parameters. An action schema specifies the preconditions and effects of all possible ground actions based on sets of literals that use the parameters of the action. The intuition is that the action schema functions as a template that specifies for each ground version of the action three sets of ground literals: one that captures the preconditions of the action, and two that capture the positive and negative effects of the action. For a ground action, the corresponding set of ground precondition literals must hold in current state in order for the ground action to be applicable. Similarly, the corresponding set of ground positive effects are added to the current state after the action is executed, while the corresponding set of ground negative effects are removed from the current state. This will become more clear in Section 3, where we give specific examples of action schemas in detail.

One way to solve STRIPS planning problems is performing state-space search. This is similar to performing a search method over a given graph, except that the nodes of the graph are implicitly specified by the available predicates and constants of the domain, while the vertices of the graph are implicitly specified by the action schemas. Similar to informed search algorithms, a forward-chain state-space planner can use heuristic functions that give an estimate about how far each state is from one that satisfies the given goal, in order to guide the search toward the most promising actions.

The STRIPS representation of states, goals, and actions as sets of literals, allows for investigating domain-independent heuristic functions, that is, functions that can be applied in any STRIPS planning domain to guide the search. The idea is that these heuristic functions exploit some aspect of the structure of the problem that can be identified just by looking at the decomposition of states to literals and the correlation of literals in the sets that specify the preconditions and effects of the action schemas. For example, a very simple domain independent heuristic function is one that counts the goal state literals that are not included in current state, and reports this number as the estimation for the number of actions that are required to reach the goal. We will refer to this heuristic function as the Goal Count heuristic (GC).

A much more sophisticated heuristic function that has proven to be very successful in many application domains is the Fast Forward heuristic (FF) [11]. This heuristic is based on building a relaxed version of the given problem by removing all negative effects from the available action schemas, and finding a solution for the relaxed problem using a simplified version of the Graphplan algorithm over planning graphs [3]. The heuristic value returned is the length of the plan that is extracted for the relaxed problem.

As a means of presenting the STRIPS domains and problems in a formal and common syntax, we will appeal to the Planning Domain Definition Language (PDDL) [9]. In PDDL a planning task is specified in two parts: i) the planning domain specifies the available predicates and action schemas, and ii) the planning problem specifies the available objects (constants) as well as the initial state and goal condition. Literals are represented in a prefix notation, e.g., literal $\textit{holding}(raw\text{-}meat)$ is written as `(holding`

Fig. 1: Screenshot of the Smart Workers RTS domain

`raw-meat)` in PDDL. Special keywords are used to specify preconditions and effects as lists of literals as we will see in the next section where we specify the Smart Workers domain using PDDL.

3 The Smart Workers domain

The Smart Workers domain follows the typical setting of real-time strategy (RTS) video games where the user manages various resources, structures, and units with the aim to eliminate all other kingdoms. Worker units are a special type of unit that are typically instructed to carry out simple tasks such as moving to places or mining for a type of resource. Such tasks are normally specified by the human player using direct commands that instruct a worker unit to perform each of the atomic actions separately.

In the Smart Workers domain the game-world is slightly more detailed in that these simple tasks also require that the worker unit uses or holds a particular resource in order to execute them. For example, in order to hunt for food the worker unit first needs to get some weapon from the armory, provided that such a structure is available. As different structures provide different tools and resources, the capabilities of the worker units depend heavily on the structures that the human player has chosen to build.

In this setting we explore the possibility that worker units are given *goals* as instructions, instead of *direct action execution commands*, and investigate how this behavior can be practically and effectively implemented. The idea is that a worker unit has an appropriate STRIPS representation of the game-world so that these advanced commands can be resolved taking advantage of methods for real-time planning. In particular, we will be focusing on one specific goal for worker units, namely bringing a food ration to the kingdom, and our intention is to identify which method is more appropriate for handling this problem given that there are various ways that this could be pursued in the Smart Workers domain. We now proceed to specify the Smart Workers planning domain using PDDL syntax.

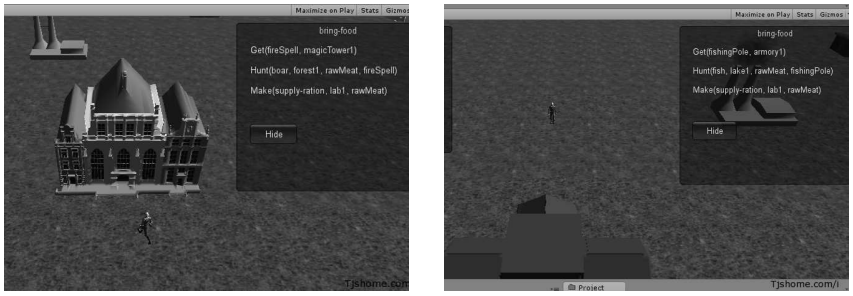

Fig. 2: Left: Plan extracted with all buildings available. Right: Alternative plan extracted with warehouse and farm disabled

3.1 The predicates of the domain

The following predicates capture the properties of the available structures, objects, and resources in an abstract way. Note that in PDDL variables are denoted with a preceding question mark.

- `(holding ?o)`: The worker unit holds object `?o`.
- `(provides ?s ?o)`: The worker unit may get or harvest object `?o` from structure `?s`.
- `(is-converted-to ?o1 ?o2)`: Object `?o1` may be converted to object `?o2` through an appropriate convert action. For instance, raw meat from hunting can be converted into a food ration.
- `(sells ?s ?a)`: Structure `?s` sells object `?o`.
- `(accepts ?s ?o)`: Object `?o` can be used in structure `?s` in order to produce a new object.
- `(lives-in ?s ?a)`: Animal `?a` is located in structure `?s`. For example, `lives-in(forest, boar)`.

We also use the following predicates to specify the type of objects.

- `(building ?b)`: Object `?b` is a building.
- `(natural-place ?n)`: Object `?n` is a mine that can be harvested for resources or a lake or a forest.
- `(weapon ?w)`: Object `?w` is a weapon.
- `(tool ?t)`: Object `?t` is a harvesting tool.

We now proceed to present the available actions in the Smart Workers domain, in the form of PDDL action schemas.

3.2 The actions of the domain

An action schema specifies the preconditions and effects of actions in terms of sets of literals that take as arguments the parameters of the action schema. The following two characterize the simple actions with which the worker unit gets an object from a building, and harvests for a resource.

```
(:action get
 :parameters (?b ?o)
 :precondition (and
   (building ?b)
   (provides ?b ?o))
 :effect (
   (holding ?o))
)
```

```
(:action harvest
 :parameters (?n ?o ?t)
 :precondition (and
   (natural-place ?n)
   (tool ?t)
   (holding ?t)
   (provides ?n ?o))
 :effect (
   (holding ?o))
)
```

For example, with the ground action `(get armory sword)` the unit can get a sword from the armory as long as the literal `(provides armory sword)` is in present in the set representing the current state. The effect of such an action would be that the fact `(holding sword)` is added to the representation of the state. For the other action schema `?o` is a type of resource, `?n` a structure, and `?t` a tool. Apart from the requirement that `?n` is of the appropriate type and provides the resource `?o`, the unit need also hold a harvesting tool. Similar to the previous action the appropriate literal is added to the representation of the state after the execution of the action.

The domain also includes actions `fish`, `hunt`, `buy`, as well as actions `craftResource`, and `craftWeapon` for creating new objects from more basic resources. These can be used in various ways to generate food rations, depending on the literals in the currents state. Some of these ways include getting rice from a windmill farm, using a weapon to go for hunting and then converting the raw-meat, producing a magic spell to go for hunting, as well as harvesting for gold, generating gold coins, and buying food from the market.

3.3 Implementation in the Unity game engine

We implemented the Smart Workers domain in the popular Unity[3] game engine, also making use of the so-called "iThink" STRIPS planner library that has been developed in the C# programming environment of Unity [1]. Note that the planner does not take PDDL code as input. Instead, the planning domain and the planning problem are specified using special functions of the iThink planner library, in a similar way as in PDDL. In particular, ground literals and ground actions are formed using directly the actual C# objects (in the programming sense) of the game-world as arguments. This allows for a more practical correlation of the actual objects in the game-world and the symbolic literals of STRIPS planning. Along the same lines, another feature of the planner is that it takes advantage of the functionality of Unity that allows to put a tag on objects of the game-world, as a means to specify a type for each object.

Also, unlike academic planners which typically perform first a pre-processing step that generates a propositional version of the problem (essentially building a structure similar to the search graph for the planning problem), iThink performs search in a textbook fashion following a closed list and an open list of visited states that are generated during the search. Some basic forward-chaining search methods and heuristics are supported, and as a means of further controlling the way that new states are generated

[3] `http://unity3d.com/`

the iThink planner provides a simple language for specifying how the search methods should generate candidates for applicable actions. This is done by means of a list of action prototypes that make use of the action schemas and the in the domain such as the following: `get-2-tag::armory-tag::weapon`. Apart from implementing the role of object types as in typed STRIPS, the order of the prototypes in the list plays a crucial role in uninformed search.

Except for the planning aspect, the Smart Workers domain was implemented as a functional mini-game. Figure 1 is a screenshot that shows part of the domain.

3.4 An example scenario

In a simple usage example of the "Smart Workers RTS" domain, initially the worker unit stands idle on the virtual kingdom waiting for a goal to be assigned to. Depending on the available buildings that the unit can use, the goal "bring a food ration" may be realized by different types of hunting as depicted in Figure 2 and Figure 2. Several types of goals may be assigned, and a vast amount of different configurations of available resources require a flexible worker behavior in order to meet the orders. For example, depending on the stage of the game only some buildings may be build already, while some others may become unavailable any time due to enemy kingdom attacks

In reality, in a real game not all workers would have this type of capability. After a certain upgrade though, worker units may also accept goals instead of direct orders, and obtain the capability of planning for a minimal depth. A second upgrade may allow some worker units to become even more flexible considering a longer search depth. A final upgrade may allow some smart workers to accept joint goals or give them the ability to order other units to perform direct commands.

4 Evaluation of planning methods in the Smart Workers domain

As mentioned earlier, we will be focusing on one specific goal for worker units, namely bringing a food ration to the kingdom, with the intention to identify which method is more appropriate for handling this problem given that there are various ways that this can be achieved in the Smart Workers domain. For this purpose we specified a series of planning problems over the Smart Workers domain, and evaluated the performance of some basic informed and uninformed state-space search methods using the planner iThink.

4.1 Problems sets of increasing levels of difficulty

The presence of specific types of buildings in the current state of the game-world allows the worker unit to perform actions that it may not be able to perform otherwise. For instance, if there is no armory in the domain the worker cannot get a weapon. Essentially, the available structures in the current state determine to a great extent the available actions of the worker unit. Following this observation we developed a series of planning problems of increasing difficulty, which differ in the types of structures that are available.

We start with Level 1 which is the easiest as all buildings are available and the goal can be reached by performing one action, and go up to Level 6 removing one type of structure each time. As the available actions of the worker unit get limited, the optimal plan becomes longer making it more challenging to identify a solution fast. Also, to investigate how the number of literals affect the difficulty of the planning problem, for each type of problem we also considered larger instances by increasing the number of buildings of each type.

4.2 Search methods considered

A usual concern among game developers against using sophisticated approaches for decision making is that it is often faster to customize the domain specification so that a simpler domain-dependent approach can be applied. In order to investigate whether this concern applies for the Smart Workers domain, we focused on some approaches that attempt to explore the spectrum between the two extremes of i) applying a sophisticated domain-independent academic method and ii) applying a domain-specific approach tailored specifically for the problem of planning for food rations in this domain.

We chose to experiment with three search methods: i) basic uninformed depth-first search (DFS) with a maximum depth limit, ii) A* heuristic search (Astar), and heuristic depth-first search (H-DFS) with a maximum depth limit, a variation of DFS which uses a heuristic function to order the nodes that are inserted into the open list in every iteration. For the informed search methods we experimented with the goal count (GC) and the Fast Forward (FF) heuristic that we mentioned in Section 2.

In order to investigate whether a domain-dependent approach would be more effective, we tweaked the action templates of the iThink planner so that action generation considers actions that are more promising first (according to our experience with the domain). In this way, the uninformed search essentially runs on what we call an "empirically optimized" action odering. We also tried the same methods with a random action ordering to examine how it affects the results.

4.3 Results

Below, we present some diagrams that compare the performances of the search methods considered with their heuristic functions, in the subsection before. Indicatively, we discuss Levels 2,3, and 4.

Level 2 In Figure 3(a), the execution time of these methods is compared, in the second level of difficulty, where the worker has to execute at least two actions to reach the goal, for example getting rice from a windmill and converting it into a food ration. By optimizing the action ordering, DFS, although a blind search method, seems to work best along with A* with the goal count heuristic (GC). Heusristic DFS goal count works also well, but methods using FF slower. This is can be explained by the amount of time needed to produce the planning graphs since each state contains many literals and numerous actions are applicable. For this difficulty of the problem, all search methods needed almost the same number of nodes to visit. The differences in execution time

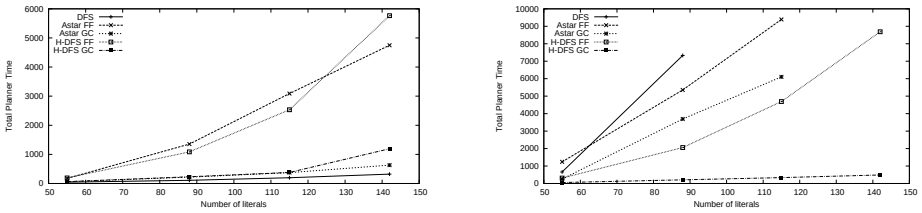

Fig. 3: Level 2 and Level 3 Total planner time in msec

had to do more with the implementation of each method, such as the sorting that A* uses, or FF mentioned before. It is interesting to note that by providing a random action ordering, the situation changes completely. DFS now is unable to return a plan in a reasonable amount of time, thus being the slowest, and algorithms that use goal count, delay greatly too, but not to the same extent as DFS. However, the FF heuristic, seems to be little affected, as expected.

Level 3 We removed some buildings from the domain of the game, resulting in fewer actions available for the worker to execute. In this level of difficulty, to reach the goal, we need at least 3 actions, meaning that the minimum depth of search is 3. A candidate optimal plan, may involve a sequence, like getting a weapon, hunting an animal in a forest for raw meat and converting it into food in the slaughterhouse. In Figure 3 we present some results about the execution time of the search methods we considered before. DFS now is the slowest to return a plan-solution, despite having provided an helping action ordering for the current goal. A* algorithm, regardless of heuristic function delays to extract solution, although the solution is optimal and being at least faster than DFS. Heuristic Depth FS proves to be the fastest one for this difficulty of the problem, especially when using the goal count heuristic, which returns a solution almost instantly and optimal. H-DFS consumes also the fewest nodes, meaning that it seems to be more accurate for the current difficulty. DFS and A* using the GC heuristic consume a large and unnecessary amount of state space, while A* using FF is more accurate, but a bit less accurate than heuristic DFS. Finally, if we alter the action ordering, DFS will be unable to return a solution in each size of the initial state, but still Heuristic Depth FS with GC function is still the fastest search method with heuristic DFS FF following.

Level 4 In order to make the current planning problem even more difficult, we removed some more buildings from the domain, limiting the actions of the worker unit and forcing him to devise more complex plans. For level 4 of difficulty, the optimal plan to satisfy the goal consists of four actions. The optimal plan is even more complicated than previous levels; the worker now has to harvest gold from a mine and produce money from it and use the money to buy a food ration from the market building. Depth First Search is still a very slow, along with A* goal count, thus being unable to return a solution in an acceptable time. Heuristic Depth FS using GC function is unable to

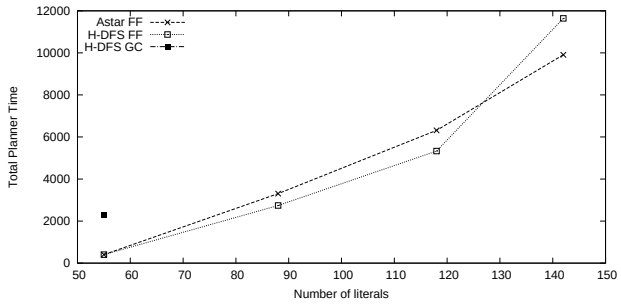

Fig. 4: Level 4 - Total planner time in msec

solve the problem, even for the smallest initial state, while in previous level was the fastest. The algorithms using FF heuristic seem to be efficient enough as they work quite well even for the largest size of the initial state where we have four buildings of each available type.

Results It is important to note how the increasing levels of difficulty mentioned before are produced by removing buildings from the planning domain. The domain may be thinned, resulting in smaller state space, but at the same time we "lose" actions that could lead directly to the goal we set. This means that the search algorithm has to look for more complicated sets of actions that could lead to the goal state, meaning greater search depth and thus larger search space consumed. For example, if we remove the warehouse from the domain, the latter will look smaller but the agent can no longer simply get a food ration and as a result, it has to find ingredients from other structures to produce one. Consequently the new plan has more than one actions.

It is also important to note that we were not able to find an optimal action ordering that will work well for DFS in all levels of difficulty. Levels such as 3 and 4 are indications as of why this is true. As we see, our "empirically optimized" action ordering fails to provide DFS with information that helps perform well. Nonetheless, we can devise another ordering that is specifically optimal for level 3 by moving up to the list those actions that are needed to solve the planning problems in Level 3. In Figure 5, we report on the running time of the different approaches with this new ordering. DFS now seems to work extremely well, even though it does not extract the optimal solution-plan.

As we anticipated though there other scenarios where this new ordering cannot cope well. In particular, in Figure 6 we report on the running time of the different approaches with this new ordering. DFS was not able to solve any problem in this Level with the new ordering.

Finally, note that search methods using the FF heuristic are not affected from the change of orderings (as expected) and that have the most consistent behavior overall.

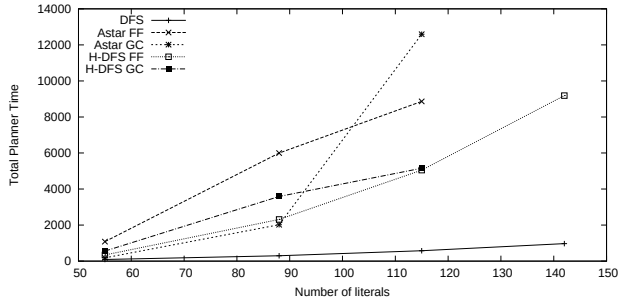

Fig. 5: Level 3 - Total planner time in msec, Optimal action ordering for Level 3

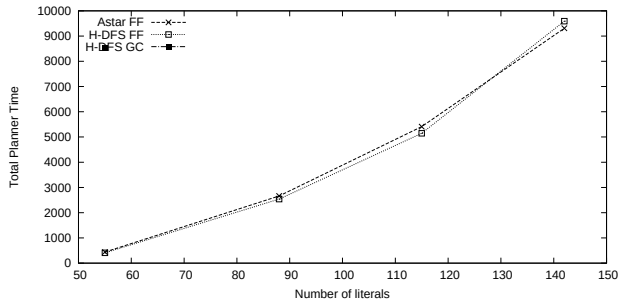

Fig. 6: Level 4 - Total planner time, Optimal action ordering for Level 4

5 Discussion

Judging from the previous results, we conclude the following. First of all, for simple problems, such as level 2 of the previous problem, simple search algorithms like depth first search can be quite efficient though the solution extracted is not always optimal. Adding also some optimizations, like altering the ordering of actions in a manner to manipulate DFS towards the solution, can render sometimes such search methods even more efficient and comparable with more sophisticated algorithms that use heuristic mechanisms. Using heuristic functions, and especially computationally involved such as FF, may prove ineffective for these simple problems as the state space is quite small. In the case of FF, it takes a lot of time to produce the planning graph given that the initial state contains numerous literals as well as numerous actions are applicable. However, as the problem difficulty increases, there is also a proportional increase in the need of using more sophisticated techniques. Simple algorithms are less efficient as their search field is vast and choose blindly the next node/state to expand, resulting often in an unacceptable execution time and quality of solution. This cannot be avoided even using domain-specific knowledge such as the optimization of action ordering.

The choice of the heuristic functions GC and FF was based on the idea of trying out both a function that is simple and easy to compute, and a sophisticated one that seems to be working well in many domains. Note also that the goal state we set, contained only one literal, meaning that goal count function would always return 1. This means, that A* combined with this function actually behaves more like Breadth First Search. A* algorithm sorts all open states according to the sum of current plan actions and the heuristic value. We conclude that, open states which are in the highest level of the state space tree are in top priority. That is why, goal count function proves ineffective when we need more and more plan actions to satisfy the goal as A* exhausts a great and unnecessary portion of the state space. FF, though, can give a more accurate estimation of the expected plan length in this domain as observed by the results.

In order for Heuristic DFS to work efficiently, it must be accompanied with a very accurate heuristic function, like FF. Using the goal count heuristic proves ineffective in many cases, especially when the difficulty of the problem increases. Even though it is classified among heuristic search algorithms, due to the fact that goal count always returns 1 value for the specific goal, it is greatly affected by the action ordering but in a completely irregular manner. For example, there are cases where this algorithm is unable to return a solution in a reasonable amount of time even when the action ordering is optimal, while with a random ordering may work quite well. This unreliability renders this algorithm with goal count heuristic inappropriate for meeting the needs of a demanding video game. However, using FF, heuristic DFS proves to be the best because as mentioned before, in most cases the function often leads straightforward to the solution, zeroing the need for backtracking.

Even when the domain of the game becomes vast, potentially due to a large number of buildings, there are various ways to simplify it though not implemented within the demo. For instance, the workers' behavior can be modified so that they can sense game objects in a specific radius instead of sensing the whole game world. Alternatively, if there are more than one buildings of each type, the worker can sense one object of each available type, in other words, providing a representative domain, of course according to what the worker is able to do or not. Consequently, by using such methods, the search domain can be simplified greatly, and judging from the diagrams in the previous section, we can always find a search method and heuristic function that work extremely well for average sizes of the domain, especially, for lower difficulties of the problem where the search depth does not exceed 5.

6 Related work

To the best of our knowledge planning methods have not been explored before in the genre of RTS video games for the purposes of guiding the worker units in the game. There are, though, other cases where planning has been used to guide the *AI opponents* in RTS games, such as for example the work of in [4]. In this approach, the authors look for means of creating an AI opponent that can produce efficiently large quantities of specific resources (food, gold, lumber) under some time constraints using a specific planner. In order to prove the effectiveness of this approach, a human extracted plan and a plan extracted by the authors' planner are put into comparison regarding time, under

the same goal of producing a specific quantity of resources. In contrast, in this work *we extend the available interactions in typical RTS domains* and focus on an agent-based approach for achieving flexible behavior for worker units in the new richer domain under the command of the human player. Note that the two approaches solve different problems.

Also, there are other cases where off-line or on-line planning has been used to guide the behavior of NPCs in other genres. A popular example of a game using such techniques is the first-person shooter game called "F.E.A.R" [17]. It uses Goal Oriented Action Planning (G.O.A.P) which is inspired by similar to classical STRIPS regarding state and action representation. Agents, being able to sense their environment decide which goal is the best to execute and plans a sequence of actions in order to achieve it. There are various other games that adopted a similar goal-oriented planning technique, including the following titles: Fallout 3, Empire: Total War, Condemned: Criminal Origins, F.E.A.R. 2: Project Origin, Deux Ex: Human Revolution, Demigod and Ghost Buster.

Another technique that is similar in spirit to classical planning is the use of Hierarchical Task Networks (HTN). In order to reach the goal, it is divided into smaller ones, easier to solve, until it is decomposed into simple actions that can be executed directly. A well-known game that uses HTN planning is the first-person shooter game Unreal Tournament [10, 15]. In Unreal Tournament, HTNs are used to coordinate the behavior of various non-player characters to execute goals collaboratively.

An interesting approach that falls in this category is the development and evaluation of an HTN planning library which used as a testbed a well known Role Playing Game, "The Elder Scrolls VI: Oblivion". This system exported as a planning solution behavior scripts for agents, scripted in the language of this game. One main the main motives of this approach was also the increasing need to produce dynamic and automated behavior for agents according to the plan extracted given that there are numerous states in which the game environment is [13]. Some other related work that has been performed in the planning community is reported in [14, 16, 2].

Finally, on a different direction the work of [6] investigates search methods for finding good action sequences for a group of units engaged in combat with enemy units. Also, [5] looks into the problem of employing search algorithms in RTS games.

7 Conclusions

In this work we were interested in extending the game play of real-time strategy video games, by providing a more realistic domain by allowing human players to instruct the worker units to perform high-level commands that involve finding an appropriate sequence of actions to execute. To realize this idea, we evaluated various STRIPS planning techniques inside a real video game environment so that we can extract conclusions with respect to which extent each method performs well and under which circumstances. We conclude that even in small problems like the ones in the Smart Worker domain, the sophisticated heuristic methods are the only ones reliable, compared to approaches we tried that focus on simple search methods and exploiting domain-dependent customizations, even though sometimes for small problems some

unnecessary overhead is added. Finally, as future work we intend to investigate how similar goals may be handled by more than one worker units in collaboration.

References

1. Vassileios-Marios Anastassiou, Panagiotis Diamantopoulos, Stavros Vassos, and Manolis Koubarakis. ithink: a library for classical planning in video-games. In *Proceedings of the 7th Hellenic conference on Artificial Intelligence: theories and applications*, SETN'12, 2012.
2. O. Bartheye and E. Jacopin. Connecting PDDL-based off the shelf planners to an arcade game. In *AI in Games Workshop at ECAI-08*, July 2008.
3. Avrim Blum and Merrick Furst. Fast Planning Through Planning Graph Analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*.
4. Hei Chan, Alan Fern, Soumya Ray, Nick Wilson, and Chris Ventura. Online planning for resource production in real-time strategy games. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2007.
5. David Churchill and Michael Buro. Incorporating search algorithms into rts game agents, 2012.
6. David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in starcraft. In *CIG*, pages 1–8. IEEE, 2013.
7. Long Edmund. Enhanced npc behavior using goal oriented action planning, 2007.
8. Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 1971.
9. M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.
10. Hai Hoang, Stephen Lee-urban, and Héctor Muñoz avila. Hierarchical plan representations for encoding strategic game ai. In *In Proc. Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05*, 2005.
11. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation, through heuristic search.
12. Wexler James. Artificial intelligence in games: A look at the smarts behind lionhead studios black and white and where it can and will go in the future. In *Spring Simulation Multiconference*, 2008.
13. J. P. Kelly, A. Botea, and S. Koenig. Offline Planning with Hierarchical Task Networks in Video Games. In *Proceedings of the Fourth International Conference on Artificial Intelligence and Interactive Digital Entertainment AIIDE-08*, 2008.
14. Buro Michael. Call for AI Research in RTS Games. In *In Proceedings of the AAAI Workshop on AI in Games*, 2004.
15. Hector Munoz-Avila and Todd Fisher. Strategic planning for unreal tournament bots. In *In AAAI Workshop on Challenges in Game AI*. AAAI Press, 2004.
16. Santi Ontanon, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. On-line case-based planning. *Computational Intelligence*, 26.
17. J. Orkin. Three States and a Plan: The AI of F.E.A.R. In *Proceedings of the Game Developer's Conference (GDC)*, 2006.
18. Megan Smith. Game ai for domination games.
19. Stavros Vassos and Michail Papakonstantinou. The SimpleFPS Planning Domain: A PDDL Benchmark for Proactive NPCs. In *Intelligent Narrative Technologies IV, Papers from the 2011 AIIDE Workshop*, volume Technical Report WS-11-18. AAAI Press, October 2011.