A FIRST LOOK AT BUILD-ORDER OPTIMIZATION IN REAL-TIME STRATEGY GAMES

Alex Kovarsky and Michael Buro
Department of Computing Science
University of Alberta, Edmonton, Alberta, Canada
email: kovarsky|mburo@cs.ualberta.ca

KEYWORDS
planning, real-time strategy games, concurrent actions

ABSTRACT

Planning is an important cognitive process. We are constantly involved in planning even when executing seemingly simple tasks such as driving to school. The ability to plan becomes essential in unfamiliar environments where we cannot rely on previously learned action sequences, but instead have to generate novel solutions by considering consequences of action choices on the fly. Planning in real-world domains poses a big challenge to current AI systems because of inadequate abstraction and search mechanisms. Human planning ability is therefore still considered superior. A good example is modern video games in which it is apparent that computer characters lack human planning, learning, and reasoning abilities. In this paper we approach the real-time planning problem by considering build-order optimization in real-time strategy games. This problem class can be formulated as a resource accumulation and allocation problem where an agent has to decide what objects to produce at what time in order to meet one of two goals: either maximizing the number of produced objects in a given time period or producing a certain number of objects as fast as possible. We identify challenges related to this planning problem, namely creating and destroying objects, concurrent action generation and execution, and present ideas how to address them.

INTRODUCTION

Planning refers to finding an action sequence that achieves a given goal. People are involved in planning for a variety of tasks in their everyday lives. Most of the time we do not realize that we are actually executing plans, since for most behaviors we use memorized action sequences. For example, we may have memorized a plan to buy groceries which could look as follows: make a list of what you need to buy, drive to the grocery store, buy what is on the list, drive home, and unload the groceries. However, what happens when one of the links in the above chain is broken? If the car does not start, we will have to plan how to achieve the goal of getting the groceries another way. We have many options of getting to the grocery store, including taking a bus, walking, fixing the car, and in turn each of those options requires planning. If a person has never taken a bus to the store before, he will need to find out about the bus's route, schedule, and fare, etc. Thus, in real-life we become engaged in planning when we are facing new situations to which we have no immediate solutions.

Today, most video game AI systems are designed as complex rule-based scripted systems. Specifically, game designers try to foresee with every possible scenario that AI characters can encounter and then write a script of actions for that situation. Such systems are difficult to maintain and expand. Moreover, it is impossible to predict and effectively plan for every potential future state. Most games do not have capabilities to handle unfamiliar situations. In such cases the AI behavior is usually ineffective and predictable. We believe that adding real-time planning capabilities to computer games will result in improved AI behavior. Recently, the commercial success of the action game F.E.A.R. (Orkin, 2005) shows the positive effects automated planning can have on the performance of AI characters. In F.E.A.R., unlike other action games, the AI system performs real-time planning that allows computer characters to adapt their behavior to the current situation.

The focus of our research is creating AI systems for real-time strategy (RTS) games such as Starcraft and Age of Empires. The common objective in RTS games is to eliminate other players through military superiority. Players first instruct workers to gather resources, then use those resources to build more workers and structures that can create military units. These are then sent to battle the enemy in real-time.

Research in RTS game AI can be divided into two branches: higher-level and lower-level AI. Higher-level AI refers to management of resources, decisions on what to build, and strategic decisions such as sending units into battle. Lower-level AI refers to the behaviors of single units and small groups of units that are given commands.

Today, most commercial RTS games are designed as rule-based systems with limited planning capabilities. However, some AI techniques such as influence maps and terrain analysis to deal with new maps have been implemented in commercial RTS games (Pottinger, 2000). Progress has also been made in wall-building (Reid and Davis, 2006), pathfinding (Sturtevant and Buro, 2005) (Demyen and Buro, 2006), and local combat (Kovarsky and Buro, 2005). The improvement of lower-level AI modules is important because without effective solutions in this area, research on higher-level reasoning and planning cannot proceed. Currently, because of the significant advances in lower-level AI the missing link for a complete RTS game AI system is a higher-level AI module responsible for global planning.

Such module could be looked at as an "all knowing" general that makes decisions affecting all global aspects of an RTS game, including resource collection, building decisions, and military expansion. Those general decisions and commands are then passed to lower-level modules which are responsible for the implementation details.

Our focus is build-order optimization in RTS games. We aim to optimize the gathering of resources and the creation of buildings and units in the initial stage of RTS games. In this research we only consider actions of our units, since in the initial game phase there is no or little interaction with the opponent. Our planning domain is defined by a technology tree that specifies the relationships between units, buildings, and resources. For example, in order to build a factory we require a worker, barracks, and sufficient resources. We consider two types of optimization problems: minimizing the time to achieve a certain goal, such as creating 2 tanks and 5 marines, or maximizing the amount of a resource or unit type in a specified time, e.g. gathering the maximum amount of iron within 10 minutes).

The build-order optimization problem presents several challenges that have not been addressed previously in planning research. One problem is the creation of new objects which can act in the world. Another is the concurrent execution of actions in RTS games, which leads to problems such as determining whether a given set of actions is executable concurrently and efficiently generating concurrent action sets.

In the remainder of the paper we first formulate a build-order optimization problem in the planning domain definition language (PDDL, (McDermott and Committee, 1998)). We then discuss the limitations of PDDL with regard to our planning domain. Subsequently, we describe the new challenges we face in build-order optimization in RTS games and then we discuss several approaches to address the unique challenges of RTS games.

```
(define (domain build-order)
  (:types worker building)
  (:predicates
    (canProduceWorkers ?b - building)
    (canProduceMarines ?b - building)
    (activatedW ?x - worker)
    (busyW ?x - worker) )
  (:functions
    (amount-of-resources)
    (num-marines) )

  (:durative-action create-worker
     :parameters (?x - worker   ?b - building)
     :duration (= ?duration 1)
     :condition (and (at start (>= (amount-of-resources) 50))
                     (at start (canProduceWorkers ?b))
                     (over all (canProduceWorkers ?b))
                     (at end (canProduceWorkers ?b)))
     :effect (and (at start (decrease (amount-of-resources) 50))
                  (at end (activatedW ?x))

  (:durative-action create-marine
     :parameters (?b - building)
     :duration (= ?duration 2)
     :condition (and (at start (>= (amount-of-resources) 100))
                     (at start (canProduceMarines ?b))
                     (over all (canProduceMarines ?b))
                     (at end (canProduceMarines ?b)))
     :effect (and (at start (decrease (amount-of-resources) 100))
                  (at end (increase (num-marines) 1))

  (:durative-action gather-resource
     :parameters (?x - worker)
     :duration (= ?duration 5)
     :condition ((at start (at ?x ?l))
                 (at start (not (busyW ?x)))
     :effect ((at end (increase (amount-of-resources) 100))
              (at start (busyW ?x))
              (at end (not (busyW ?x)))))
```

Figure 1: Simple build-order domain Specification in PDDL with three actions: create-worker, create-marine, and gather-resource.

BUILD-ORDER OPTIMIZATION AND PDDL

Research on automated planning has mostly concentrated on classical planning, which in short can be summarized as planning without regard to time. In such planners a plan is an ordered sequence of actions, actions are instantaneous and do not interact with each other. However, in real life very few domains adhere to such restrictions. Therefore, in recent years temporal planning has gained attention. Temporal planners take time into account. Each action requires a certain time to execute and in certain situations several actions are allowed to execute concurrently.

The development of PDDL and subsequently the PDDL 2.1 (Fox and Long, 2003) extension, that incorporates time in its semantics, have further facilitated the research for creating temporal planners. However, the PDDL 2.1 semantics is still too restrictive for our RTS domain. But since PDDL has become a standard in the planning research community, we would like to stay compatible with PDDL.

PDDL was developed to standardize planning domain and problem description in order to enable different planners to compete against one another in interna-

```
(define (problem SimpleBuildOrder)
  (:domain build-order)
  (:objects
     commandCentre barracks - building
     worker1 worker2 worker3 worker4 - worker )
  (:init
     (not (activated worker1))
     (not (activated worker2))
     (not (activated worker3))
     (not (activated worker4))
     (canBuildMarines barracks)
     (canProduceWorkers commandCentre)
     (= (num-marines) 0)
     (= (amount-of-resources) 700) )
  (:goal (and (>= (num-marines) 5)))
  (:metric minimize (total-time)))
```

Figure 2: Build-order problem specification in PDDL. Starting with two buildings, the goal is to create 5 marines as fast as possible using at most four workers.

tional planning competitions. PDDL supports the syntax of STRIPS, ADL, and some other previously used planning languages. A PDDL definition of a planning problem consists of two parts: the domain definition and the problem definition. The domain definition file is where the types, predicates, functions and actions are defined, while the problem definition file is where the objects of types defined in the domain files are declared, the predicates and functions acting on objects are initialized, and the goal conditions for the plans are specified. The following example shows a simplified PDDL RTS domain and problem files.

In the domain file in Figure 1 there are two types (worker and building), four predicates, and two numerical functions. Functions in this context are used to store numerical fluents such as resource amounts. The two actions are temporal actions for which the duration clause specifies the number of steps required for the action to complete. In the condition clause the preconditions for triggering the action are specified. The “at start”, “over all” and “at end” expressions specify when a given condition has to hold (i.e. at the beginning of action execution, during, or at the end, respectively). In the effect clause, the effects of a given action are specified. The effect on functions is a numerical change of functions values (increase or decrease), while the effect on predicates is Boolean, i.e. a predicate for a given object or objects can be set to true (e.g. ActivatedW ?x), or to false (not (ActivatedW ?x).

The first action in Figure 1 is create-worker. In order for this action to execute the amount-of-resources value has to be at least 50 and the building has to be able to produce workers (i.e. CanProduceWorkers is true). The effect of this action is a reduction in resources and the activation of a worker (i.e. activatedW is true). The second action in Figure 1 is create-marine. Similarly, in order for it to execute there must be sufficient resources and the building has to be able to produce marines. The effect is a reduction in resources and numerical increase in the number of marines. Unlike workers, marines are not objects. They are modeled numerically, like resources. We do not have to model them as objects since in the domain in Figure 1 they are not involved in any actions. Workers have to be created as objects since workers are involved in the gather-resource action.

In the problem specification in Figure 2 two objects of type building (i.e. commandCentre and barracks) and four objects of type worker are declared. These objects as well as functions are initialized in the :init clause. The worker type objects are set to not activated (since they are not built yet) and the barracks and commandCentre objects are enabled to build marines and workers, respectively. The goal (:goal) of this plan specification is to increase the number of marines to 5 and to do so in a minimal possible time (:metric).

PLANNING IN RTS ENVIRONMENTS

PDDL is quickly becoming the standard input language for planners (Kautz and Selman, 1999) (Hsu et al., 2006). The International Planning Competition (IPC) is an annual event, which is run in conjunction with the ICAPS conference. Currently, IPC is the key test-bed for both classical and temporal planners. However, even the newest extension of PDDL fails to address some of the challenges present in RTS games. First, we divide the issues arising in build-order optimization into two parts: object creation and destruction and concurrent action execution. We then discuss each of the problems, describe the restrictions of the PDDL semantics with respect to each problem and propose ways to relax these restrictions.

Object Creation and Destruction

The typical problems that PDDL aims to address and that are used in the international planning competition are so called closed problems in which the number of objects in the world remains constant. Even the most recent version of PDDL does not allow for the creation or deletion of objects. In RTS games, however, creating objects is key. A typical game starts with a small number of units and a limited amount of resources. Then, those units can create structures that can produce new units, mine resources, or perform other functions. Another important aspect of RTS games is combat. As the RTS game progresses, military units start fighting with their opponents. As a result some units or structures can be destroyed. Currently, PDDL does not provide means for object destruction. Even though no explicit mechanisms for the creation and the destruction of objects exist there is a way to implicitly simulate object creation and destruction in PDDL. All objects that can potentially be created in the future have to be specified in advance in the problem file. Creation and destruction of an object in this setting means switching this object on and off by using a predicate (for example activate (object name)). Refer to Figure 1, where a worker is created by activating (activatedW ?x) a previously defined (Figure 2) object (e.g. worker1) from the problem definition file. This approach is computationally inefficient and awkward to implement. Since all objects have to be specified in advance, at every point the planner has to examine each of the objects when generating possible actions. This examination is inefficient, since in the beginning of the planning process most of the objects are not active. Furthermore, in some scenarios it is difficult to predict the maximum number of objects that are needed in advance. Pre-declaring a large number of objects will lead to the above-mentioned inefficiency, while declaring too few objects might result in a shortage of objects (and possibly in an inability to achieve a given goal). The problem of object creation and destruction should be straightforward to address. We can add an explicit object CREATE and DESTROY capability to PDDL. This can be done by adding a new effect to any given action that will create or destroy a certain object by adding or removing that object from the list of objects. For example, in the domain specification in Figure 1 instead of activatedW worker, we would have a new object create clause. In Figure 2 we will no longer have to specify the worker objects in advance.

Concurrent Action Execution

RTS games are inherently concurrent environments. In many situations objects can execute their actions simultaneously. Thus, when we have a set of actions, we need to determine whether such a set is executable concurrently. Concurrent executability depends on interdependence among actions. If all actions in a set are independent of each other, then they all can be executed concurrently. Sometimes, however, actions in RTS games are dependent. For example, building actions usually require resources, while resource gathering actions produce resources (e.g. create-marine and gather-resource, respectively). Whether a set consisting of resource producing and consuming actions is executable simultaneously depends on the accumulated resource amount, the amount currently produced, and the amount being consumed. Such computation is non-trivial in general. In recent years, some progress has been made on concurrently executing actions with shared resources (or numeric fluents) (Lee and Lifschitz, 2001) (Erdem and Gabaldon, 2005). This research has concentrated on formalizing the semantics for such actions, without much emphasis on the computational effort of generating concurrent action sets and determining concurrent action executability. Another challenge is generating all sets of concurrent actions efficiently given all actions that can be executed individually.

To understand the computational effort required for determining whether a set of concurrent actions is executable, first we need to make a distinction. Two types of concurrent actions are possible: serializable and non-serializable. By serializable we refer to the sets of actions, which when executed serially, one after another, have the same result as when executed concurrently. Non-serializable actions produce different effects. For example, two units may be required to lift an object or two actions are interlocked in such a way that both preconditions require the effect of the other action to be true. For typical RTS game build-order optimization problems considering serializable actions is sufficient.

A second distinction relates to the action execution environment. In the first case one could demand that for any chosen execution sequence the preconditions of all actions in the sequence is met. Alternatively, a set could be called serializable if only one such sequence exists. This approach can lead to faster plans, since more sequences will be declared serializable. However, the first condition is more robust, since in some environments there is no way to ensure that a set of actions will be executed in a given order. This is especially true for RTS game engines.

PDDL only allows for limited concurrency. No two or more actions can simultaneously use a given resource if at least one of the actions is changing its value. In PDDL actions can be executed concurrently as long as they are independent from each other. This is the so-called "no moving targets" rule which is very conservative and will prevent resource related actions from being executed simultaneously in our RTS game environment. For instance, in Figure 1, actions create-worker and create-marine consume the same type of resource. Assuming that their preconditions are satisfied (i.e. there are enough resources to create both unit types), these two actions will not be allowed to execute concurrently in PDDL because they both modify the same resource. Thus, in order for PDDL to work properly in our domain we need to remove the "no moving targets" restriction. This relaxation allows for dependent concurrent actions but also increases the computational effort for checking concurrent executability of an action set and for generating concurrent sets of actions.

Checking Concurrent Executability

One way of reducing the computational effort of planning in RTS environments is through examining the types of actions required (i.e. the expressiveness of their preconditions and effects) for a typical problem specification. By restricting the complexity of preconditions and effects we can greatly improve the runtime speed of determining whether a given set of actions is executable concurrently. We have started to look at constructing a hierarchy of precondition and effect restrictions with growing expressiveness that still allows us to decide robust serializability quickly. The first level of the hierarchy is when all actions are independent of each other.

Therefore, all actions can be executed concurrently if all the preconditions hold. At the second level the actions are no longer independent of each other. In the RTS domain this means that two or more actions can increase and decrease the value of a single resource. At this level we restrict the preconditions to having only logical operators (i.e. no arithmetic operators) and effects to having only commutative operations (i.e. increase and decrease but not set). Here the runtime effort will partly depend on the available amount of the shared resource. At level three of the hierarchy we will allow for a number of resources to be shared between actions. At the next level we will increase the expressiveness once more by allowing preconditions to contain arithmetic operators. Finally, we will examine the general case in which effects are not commutative.

Generation of Action Sets

Another challenge is to generate all possible sets of concurrent actions efficiently. Given the set of actions executable at a certain time point, we need to generate potential sets of such actions efficiently. We should aim towards complexity that is linear with respect to the number of generated sets of actions. In general, given n possible actions there are 2^n ways to choose sets of actions of arbitrary length. Such computation is infeasible in a real-time environment. Thus, suitable approximations need to be found. We could allow objects to execute at most one action at a time. The number of generated action sets will then be reduced since actions performed by the same object can be grouped together and will be left out from further consideration once a single action from that group is selected in a potential action set. Another way to decrease the number of potential action sets is through abstraction. The main challenge for planning in RTS games is that the game is unbounded — meaning the number of units (i.e. workers, buildings, military units) generally grows as the game progresses. As the number of units reaches a certain point, generating all potential action sets will become infeasible in real-time. In most RTS games, however, several units are often assigned to perform similar tasks. Thus, one reasonable approach is to group units by their type into groups, which will be treated as "super-units" and only execute identical tasks during the planning process. Such groups can be flexible. That is, new objects can be added and removed to groups and groups themselves can be merged to form larger groups, or split into smaller groups. How to do such grouping to maximize the use of available computational resources is an interesting research issue by itself. Yet another way of decreasing the number of actions is forcing units to perform a given task for a certain minimum number of time steps. Again this will result in a lower number of units available at a given time and will reduce the number of potential action sets, at the cost of optimality.

CONCLUSION

In this paper we have introduced the build-order optimization problem for real-time strategy games and discussed the following core challenges for creating an automated planning system for this domain: how to deal with object creation and destruction in PDDL, how to decide what actions can be executed simultaneously, and how to generate action sequences efficiently? We have presented initial research ideas on how to tackle these problems. Our hope is that these become the seed for a high-performance RTS game planning system that can be used to aid human players and to improve the playing strength of computer opponents.

ACKNOWLEDGMENTS

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

Demyen D. and Buro M., 2006. *Efficient Triangulation-Based Pathfinding.* In *Proceedings of the AAAI Conference.* Boston, 942–947.

Erdem E. and Gabaldon A., 2005. *Cumulative effects of concurrent actions on numeric-valued fluents.* In *Proceedings of the AAAI Conference.* Marina del Ray, 627–632.

Fox M. and Long D., 2003. *PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence.*

Hsu W.; Wah B.; Huang R.; and Chen Y., 2006. *Handling Soft Constraints and Preferences in SGPlan.* In *Proceedings of ICAPS Workshop on Preferences and Soft Constraints in Planning.*

Kautz H. and Selman B., 1999. *Unifying SAT-based and Graph-based Planning.* In *Proceedings of International Joint Conference on Artificial Intelligence.* Stockholm.

Kovarsky A. and Buro M., 2005. *Heuristic Search Applied to Abstract Combat Games.* In *Proceedings of the Eighteenth Canadian Conference on Artificial Intelligence.* Victoria.

Lee J. and Lifschitz V., 2001. *Additive Fluents.* In *Proceedings of the AAAI Spring Symposium: Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning.* 116–123.

McDermott D. and Committee A., 1998. *PDDL – the planning domain definition language.* In *Technical Report.*

Orkin J., 2005. *Agent Architecture Considerations for Real-Time Planning in Games.* In *Proceedings of the AIIDE Conference.*

Pottinger D., 2000. *Terrain Analysis in Realtime Strategy Games.* In *Computer Game Developers Conference.*

Reid T. and Davis I., 2006. *AI Base Building in Empire Earth II.* In *Proceedings of the AIIDE Conference.* Marina del Rey, CA.

Sturtevant N. and Buro M., 2005. *Partial Pathfinding Using Map Abstraction and Refinement.* In *Proceedings of the AAAI Conference.* Pittsburgh, 1392–1397.