Extending Online Planning for Resource Production in Real-Time Strategy Games with Search

Hei Chan, Alan Fern, Soumya Ray, Nick Wilson and Chris Ventura
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97330
{chanhe,afern,sray,wilsonic,ventura}@eecs.oregonstate.edu

Abstract

Planning in domains with temporal and numerical properties is an important research problem. In prior work, we presented an online planning architecture for resource production problems in RTS games. At every decision point, our planner constructs plans that achieve a certain fixed set of intermediate renewable resource goals. It then uses the plan with the smallest makespan to choose an action at the current decision point. While each plan it considers is suboptimal, we showed empirically that this action selection strategy was competitive with human players in solving large resource goals. In this work, we investigate the effect of reducing one source of suboptimality in the plan generation step. Instead of considering a fixed set of intermediate resource goals, we allow the planner to search over a variable but bounded set of goals. We investigate empirically the plan quality of three different problems for our planning architecture.

Introduction

An important component of real-time strategy (RTS) games, such as Warcraft, is the problem of fast resource production. In resource production, the player has to produce (or gather) various raw materials, buildings, civilian and military units, to improve its economic and military power. A typical RTS game usually involves an initial period where players rapidly build their economy via resource production, followed by military campaigns where those resources are exploited for offense and defense. Thus, winning the resource production race is often a key factor in overall success.

In our previous work (Chan *et al.* 2007), we have focused on automated planning in the RTS resource production domain. In particular, we have developed an action selection mechanism that can achieve any reachable resource goal as quickly as possible. Such a mechanism is useful as a component for computer RTS opponents and as an interface option to human players, where a player need only specify what they want to achieve rather than manually orchestrate the many required low-level actions. In addition to the practical utility, RTS resource production is interesting from a pure AI planning perspective as it encompasses a number of challenging issues. First, resource production involves temporal actions with numeric effects. Second, performing well in this task requires highly concurrent activity. Third, the real-time constraints of the problem require that action selection be computational efficient in a practical sense. Unfortunately, most existing planners are not applicable to this domain either because they do not handle both time and numbers or they are simply too inefficient to be useful.

The action selection mechanism we have developed is based on an online planning architecture. At each decision epoch, we use a combination of means-ends analysis and heuristic scheduling to quickly generate satisficing plans, each of which achieves a chosen intermediate goal. We use the plan with the shortest makespan to select actions at the current decision epoch. While each plan we consider is suboptimal, we showed empirically in prior work that the resulting action selection strategy was competitive with human players in achieving large resource goals.

The choice of which intermediate goal to achieve is central to the success of our planning architecture. In resource production, there are often units that, if produced, increase the rate at which resources can be harvested. Production of such units is usually not *necessary* to achieve the goal, though they may decrease the makespan of the resulting plan. Further, since an unbounded number of such units can be produced, searching over the space of all units is usually infeasible. In experiments with many real-world planners, we found that no planner was able to successfully produce plans that created intermediate units of this sort, even when they would have vastly reduced the makespan. In our work, production of such units are the intermediate goals that we explicitly plan for while achieving the final resource goal.

To keep the search computationally manageable, in prior work we only considered a fixed set of intermediate goals. In particular, at each decision epoch, we only considered building one extra unit of each type of renewable resources. While we observed empirically that this worked well, it is possible that we could achieve even better performance with a more exhaustive search through combinations of units. Obviously, the planning time will increase when we search over this larger set of unit combinations. Due to the online setting of the RTS game, we will need to impose a limit on the search depth and/or search time. We will show empirical results on three types of problems, with varying degrees of difficulties, in our RTS resource production domain.

Figure 1: A screenshot of Wargus.

The RTS Resource Production Domain

The two key components of the RTS resource production domain are resources and actions. Here, we define resources to include all raw materials, buildings, civilian and military units. While the player can reason about each individual object at a lower level, we will reason about them at a higher level by aggregating them into their various types and deal with each of the total numerical amounts in the game state. While this abstraction will lead to a certain amount of sub-optimality, it greatly aids in making the planning problem more manageable, and as our experiments demonstrate still allows for high quality plans. As an example RTS game, and the one used in our experiments, Figure 1 shows a screen shot of the RTS game Wargus. At the current game state, the player possesses a "peasant", which is a type of civilian worker unit, and a "townhall", a type of building. A peasant may collect gold by traveling to the gold mine, then returning to the townhall to deposit the gold, or it may collect wood by traveling to the forest, then returning to the townhall to deposit the wood. When enough gold and wood are collected, a peasant may also build certain buildings, such as "barracks". Barracks may then be used to create "footmen", a type of military unit, provided that other resource preconditions are met.

Human players typically have no difficulty selecting actions that at least achieve a particular set of resource goals. However, it is much more difficult, and requires much more expertise, to find close to minimal makespan plans. As an example consider the seemingly simple problem of collecting a large amount of gold starting with a single peasant and townhall. One could simply repeatedly collect gold with the single peasant, which would eventually achieve the goal. However, such a plan would be far from optimal in terms of time-to-goal. Rather, it is often faster to instead collect gold and wood for the purpose of creating some number of additional peasants (which consumes gold and wood) that will subsequently be used to collect gold concurrently and hence reach the resource goal faster. In practice it can be quite difficult to determine the correct tradeoff between how many peasants to create, which require time and resources, and the payoff those peasants provide in terms of increased production rate. The problem is even more difficult than just described. For example, one must also provide enough supply by building "farms" in order to support the number of desired peasants and footmen. This requires even more time and resources. One could also consider building additional townhalls and barracks, which are used to create peasants and footmen respectively, to increase their rates of production. Our online planner attempts to approximately optimize these choices while maintaining computational efficiency.

For our experimental testbed we selected Wargus because it has common properties with many popular RTS games and it is based on a freely available RTS engine. We will now review the properties of RTS resource production that are crucial to our design of the planning architecture.

At any time, a player can choose to execute one or more actions, defined from the action set of the game. Each action produces a certain amount of products, but also consumes a certain amount of other resources, and requires that some preconditions are met before it can be executed. Actions are usually durative, i.e., they take a certain amount of time to finish upon which the products are added to the game state. In RTS games, resource-production actions are usually deterministic, and the preconditions, effects, and durations of each action are usually given or can be easily discovered through game-play. For certain actions, where a unit has to travel to a destination for an action to take place, the duration of the action will vary due to the spatial properties of a game map. However, for simplicity we assume we have a constant duration for each instance of an action. On average over the many actions taken during a game, this turns out to be a reasonable assumption. We note that extending our approach to incorporate durations that are functions of the current state is straightforward.

In our representation, the game state at time t consists of: (1) for each resource R_i, the amount r_i possessed by the agent and (2) the list of actions $A_i, i = 1, \ldots, m$ currently being executed along with the start and end times t_i^s and t_i^e for each ($t_i^s < t < t_i^e$). We refer to the state reached when all actions currently executing in S have terminated as the *projected game state*, denoted by $Proj(S)$. This state is timestamped with $t = \max_{i=1,\ldots,m} t_i^e$, has resources updated according to the effects of the actions A_i, and no actions being executed.

The objective of a player in this domain is to reach a certain resource goal, $G = \{R_1 \geq g_1, \ldots, R_n \geq g_n\}$, defined as constraints on the resources, from the current game state. Often, many of the constraints will be trivial, ($R_i \geq 0$), as we may only be interested in a subset of resources. To achieve G, a player must select a set of actions to execute at each decision epoch. These actions may be executed concurrently as long as their preconditions are satisfied when they are executed, as the game state is changed throughout the course of action. In essence, the player must determine a plan, which is a list of actions, $((A_1, t_1^s, t_1^e), \ldots, (A_k, t_k^s, t_k^e))$, where A_i is an action that starts at time t_i^s and ends at time t_i^e. While this domain

does not require concurrency to achieve the goals in a formal sense (Cushing *et al.* 2007), plans with short makespan typically involves a large amount of concurrency.

In Wargus, and many other RTS games, each precondition and effect is specified by providing the name of a resource, an amount for that resource, and a usage tag that specifies how the resource is used by the action (e.g. shared, consumed, etc). We define four possible resource tags:

- `Require`: An action requires a certain amount of a resource if it needs to be present throughout the execution of the action. For example, the collect-gold action requires the presence of a townhall. In this case, the same townhall can be used for concurrent collect-gold actions, as the townhall is not "locked up" by the collect-gold actions. Thus, the requires tag allows for sharing of resources.
- `Borrow`: An action borrows a certain amount of a resource if it requires that the resource amount be "locked up" throughout the execution of the action, so that no other action is allowed to borrow those resources during its execution. After the action has completed the resource amount is freed up for use by other actions. For example, the collect-gold action borrows a peasant. During the execution of the collect-gold action, the borrowed peasant may not be borrowed by any other action. After the collect-gold action is finished, the peasant becomes available again and can be used for other actions. Therefore, to allow concurrent collect-gold actions, multiple peasants must be used.
- `Consume`: An action consumes a certain amount of a resource at the start of its execution, as this amount is deducted from the game state. As the game state must obey the constraint that every resource value is non-negative, the inferred precondition of the action is that this resource amount must be present at the start of the action. For example, the build-barracks action consumes 700 units of gold and 450 units of wood.
- `Produce`: An action produces a certain amount of a resource at the end of its execution, as this amount is added to the game state.

These tags are similar to resource requirement specifications used in the scheduling literature (for example, see (Ghallab, Nau, & Traverso 2004)). Given the above tags, Figure 2 gives the definitions of a subset of the resource-production actions in Wargus. In future work, we plan to consider extensions to this specification, for example, by considering consume and produce tags that specify rates of consumption or production, or allowing resource consumption to happen at the end of an action.

Note that we could have used a more traditional domain specification language such as PDDL2.1 (Fox & Long 2003) to describe our domain. However, for this work we choose the above representation to make the key resource roles explicit, which will be leveraged by our algorithm. Figure 3 shows two actions encoded using PDDL2.1. It is fairly straightforward to translate any action described by the keywords above into PDDL. Further, it is likely that the roles played by the `require`, `borrow`, `consume` and `produce` tags could be automatically inferred from a restricted subclass of PDDL.

```
resource gold
resource wood
resource supply
resource townhall
resource barracks
resource peasant
resource footman

action collect-gold :duration 510
 :require 1 townhall :borrow 1 peasant
 :produce 100 gold
action collect-wood :duration 1570
 :require 1 townhall :borrow 1 peasant
 :produce 100 wood
action build-supply :duration 620
 :borrow 1 peasant :consume 500 gold 250 wood
 :produce 4 supply
action build-townhall :duration 1530
 :borrow 1 peasant :consume 1200 gold 800 wood
 :produce 1 townhall
action build-barracks :duration 1240
 :borrow 1 peasant :consume 700 gold 450 wood
 :produce 1 barracks
action build-peasant :duration 225
 :borrow 1 townhall :consume 400 gold 1 supply
 :produce 1 peasant
action build-footman :duration 200
 :borrow 1 barracks :consume 600 gold 1 supply
 :produce 1 footman
```

Figure 2: Resource and action specification of the simplified Wargus domain.

Given the action specifications, we divide the resources into two classes, renewable and consumable resources. Consumable resources are those that are consumed by actions, such as gold, wood, and supply (a peasant or footman cannot be built unless there is an unused supply). Renewable resources are those that are required or borrowed by actions, such as peasants, townhalls and barracks. Generally, a resource is either renewable or consumable, but not both, and this can be easily inferred from the domain description. We observe that multiple renewable resources are usually not essential to achieve any given resource goal, since most actions borrow or require only one of such resources. However, if multiple such resources are available, they can vastly reduce the makespan of a plan by permitting concurrent actions.

To recap, some key properties of our domain are:

1. Actions have durations;
2. There are multiple units, so actions can be executed concurrently;
3. Units and buildings can be created as the game progresses;
4. Many actions involve numeric fluents;
5. Solution plans typically involve a large number of actions compared to most standard planning benchmarks, and;
6. In our setting, the planner must find a plan in real-time.

```
(:durative-action collect-gold
  :parameters ()
  :duration (= ?duration 510)
  :condition
   (and (over all (> total-townhall 0)))
        (at start (> avail-peasant 0))
  :effect
   (and (at start (decrease avail-peasant 1))
        (at end (increase avail-peasant 1))
        (at end (increase total-gold 100))
        (at end (increase time ?duration))))

(:durative-action build-townhall
  :parameters ()
  :duration (= ?duration 1530)
  :condition
   (and (at start (> avail-peasant 0)))
        (at start (>= total-gold 1200))
        (at start (>= total-wood 800))
  :effect
   (and (at start (decrease avail-peasant 1))
        (at start (decrease total-gold 1200))
        (at start (decrease total-wood 800))
        (at end (increase avail-peasant 1))
        (at end (increase total-townhall 1))
        (at end (increase avail-townhall 1))
        (at end (increase time ?duration))))
```

Figure 3: PDDL2.1 specification of the collect-gold and build-townhall actions.

Thus, our domain exemplifies some of the hardest aspects of planning. Recent research has resulted in several planners that are capable of handling some of these aspects. Examples include SAPA (Do & Kambhampati 2003), MIPS-XXL (Edelkamp, Jabbar, & Nazih 2006), SGPlan (Chen, Wah, & Hsu 2006), LPG and LPG-td (Gerevini, Saetti, & Serina 2006), and TM-LPSAT (Shin & Davis 2005). However, experimental results have showed that none of them could satisfactorily solve the RTS resource production problem (Chan *et al.* 2007).

In spite of the above, certain properties of our domain specification help us to efficiently create satisficing plans. First, the dependency structure between resources is such that, if the initial state has a townhall and a peasant (and assuming the world map has enough consumable resources like gold and wood), there always exists a plan for any resource goal. Further, if such a state cannot be reached, no such plan exists. Thus, we focus our attention to initial states with at least these elements. In Wargus, and other RTS games, it is straightforward to hand-code a scripted behavior to reach such a state if the game begins in a state without the required elements, after which our automated planner can take over with the guarantee of computational efficiency. Second, we observe that the amount of renewable resources in a problem never decreases, since no unit is destroyed in our scenarios. Third, by the Wargus action specification, all effects at the start of an action are subtractive effects, while all effects at the end of an action are additive effects. Fourth, again by the Wargus specification, for each resource, there is exactly one action that produces it. This property implies that every plan that produces the goal resources from a game state must contain the same set of actions (though possibly not in the same *sequence*). Conversely, suppose we have two executable plans from the same state consisting of the same set of actions, but with different starting times for some of the actions. Then the final game states after executing the two plans will be the same. This is due to the property of commutativity of action effects, as the game state is changed by the increase or decrease of resources according to the actions in the Wargus domain. Each of these properties is used by our planner to search for satisficing plans more efficiently. Each property can be relaxed, but would result in a less efficient search process.

Review of Planning Architecture

In this section, we review the architecture of our online planner (Chan *et al.* 2007). An online planning architecture is suitable for the RTS setting where goals and environments change over time. To adapt to such changes, our planner replans every decision epoch using the current goal and game state. To find a new plan, it carries out a bounded search over possible intermediate goals. The set of possible intermediate goals includes all states that have an extra renewable resource of every type. For each such goal, the planner employs a sequential planner followed by a heuristic scheduling process to generate a plan to reach the overall goal via the intermediate goal. To select an action to be executed, the planner chooses the plan with the smallest makespan. If this plan has any action that is executable at the current game state, that action (or actions) is started. Notice that the plans generated by the planner are not usually completely executed—when the planner replans at the next decision epoch using the game state at that point, it may not obtain a suffix of the plan it found at the current epoch. However, constructing such plans are valuable because they help in action selection at the current step.

We now briefly review the two components of our online planner. The first component is a sequential planner which outputs a sequential plan to achieve a given goal from a given initial state. In principle, any off-the-shelf sequential planner can be used in this step. However, given the specific properties of our domain discussed earlier, a simple sequential planner based on means-ends analysis (MEA) (Newell & Simon 1995; Fikes & Nilsson 1971) suffices. MEA operates by selecting a subgoal to solve which will decrease the difference between the initial state and the goal state, and then executing the necessary actions to solve the subgoal. Then from the new state which satisfies the subgoal the process is recursively applied until we reach the goal state. We can show that this procedure always results in a plan containing the minimum number of actions to the goal state.

The second component of our online planner is a heuristic scheduler. To accurately estimate the utility of any renewable resources, we need to reschedule actions in the sequential plan found above to allow concurrency and decrease the makespan. We do this by using a heuristic scheduling procedure that traverses the found action sequence in order. For each action A_i, the procedure moves the start time of A_i

to the earliest possible time such that its preconditions are still satisfied. Assume that A_i starts at time t_i^s, and the state $R^+(t_i^s)$ is the resource state at time t_i^s after the effects of all actions that end at time t_i^s are added to the game state, and $R^-(t_i^s)$ is the resource game state before the effects are added. Obviously, the preconditions of A_i are satisfied by $R^+(t_i^s)$. If they are also satisfied by $R^-(t_i^s)$, this means the satisfaction of the preconditions of A_i is not due to any of the actions that end at time t_i^s, and we can now move action A_i to start earlier than t_i^s, to the previous decision epoch (time where an action starts or ends). This is repeated until the preconditions of A are satisfied by some $R^+(t^s)$ but not $R^-(t^s)$, i.e., the satisfaction of the preconditions of A is due to the actions that end at time t^s. The plan is now rescheduled such that action A starts at time t^s, and we can proceed to attempt to reschedule the next action in our sequential plan. It can be shown that this procedure results in a sound concurrent plan.

Given the two components described above, our planner constructs a plan to achieve the resource goal from the current state via each intermediate goal. The (concurrent) plan with the smallest makespan is then used to select an action at the current state.

Searching Over Intermediate Goals

A key element of our planning architecture is the explicit search over a set of intermediate goals while solving the final resource goal. The rationale for this is as follows. It is clear that we can easily find a successful plan which has a minimum number of actions and creates the minimum amount of renewable resources, such as peasants. However, creating additional renewable resources can decrease the makespan of a plan (even though this new plan now has more actions), if the time penalty paid by creating these resources is compensated by the time saved by the concurrent actions allowed by the additional renewable resources. This step is never explicitly considered by many planners, or the plans become too complex if an unbounded search over all possible intermediate goals is considered. To get around this problem, in prior work, we explicitly found plans which achieved the intermediate goal of creating an additional fixed unit of renewable resources, such as an additional peasant, then found a plan which achieved the goal from this intermediate goal state. The two plans are then combined into a single plan, and we check if the new plan has a shorter makespan than the original plan. If so, we prefer the new plan which produces the additional renewable resources.

Clearly, the procedure outlined above may be suboptimal, because resources are often subject to threshold effects: while producing x or less does not decrease the makespan, producing more than x does. For example, consider the subgoal of creating one peasant in Wargus. It can only be created when there is an unused supply. Therefore, when there is no unused supply, a precondition of creating an additional peasant is to build a farm, which provides supply for four additional units. Thus, it may be possible that creating only one additional peasant may not shorten the makespan of the plan, because the payoff of one additional peasant cannot overcome the reduction in resources when building the farm. However, creating more than one additional peasant may reduce the makespan. In the present paper, we implement a search procedure that systematically explores different resource combinations. This procedure may thus discover in the situation above that the best intermediate goal is to build multiple additional peasants.

The search procedure that we implement is a bounded best-first heuristic search over possible intermediate goals. In particular, for each renewable resource, we set an upper bound that seems reasonable according to the domain. For example, while it seems reasonable to consider five additional peasants as an intermediate goal, it usually does not make sense to consider five additional townhalls. So for any search iteration, we set the upper bound on the number of peasants to be larger than the upper bound on townhalls. We can also set a time bound to limit our search. This is necessary due to the online setting of RTS games.

Given the bounds on search depth and time, we can employ a standard best-first search approach, using the makespan of the found plans as the heuristic evaluation function. Consider a search tree where each node represents an intermediate goal, and a set of search operators which add one additional resource of each type. The root of the search tree is a plan without any intermediate goal. We first expand this node, and consider the intermediate goals of only one additional resource of each type, and call the planner to find a plan for each of these intermediate goals. We use the makespan of the plan as the heuristic value of the node. From these nodes, we expand the one with the shortest makespan, by adding one additional resource of each type to the chosen intermediate goal. Note that even if none of these new plans improves on the makespan of the original plan, the search continues.

In summary, at each iteration of expansion, we choose from the unexpanded nodes the one with the shortest makespan (assume that the intermediate goal represented is G'). We expand it by adding one additional resource of each type to G', and call the planner to find a plan for each of these new intermediate goals. Note that our online planner replans for each new intermediate goal independently, and not from the plan for G'. If at any point we reach the upper bound assigned to a resource, that resource is not increased further, or if we reach the time limit, we terminate the search. From all the plans produced throughout the search, we can choose the plan with the shortest makespan, and execute actions suggested by this plan.

There are three features we can implement to make the search more computationally efficient. The first feature is a mechanism which checks whether any of the newly created nodes have been visited before, by maintaining a memory of all previously visited nodes. If the node has been visited, we do not need to call the planner again to produce a redundant plan.

The second feature is to consider search operators which add multiple extra renewable resources. For example, when we expand a node, instead of considering just the operator of adding one additional peasant, we can also consider the new search operator of adding multiple additional peasants (for example, four additional peasants, given that a new farm

can support four peasants). This is because we expect the best node to have an intermediate goal of creating many additional peasants, and by considering this new search operator, we can reach this node much quicker in the search. This is useful when the search is given a tight time limit.

The third feature is that instead of finding a plan which satisfies a singular intermediate goal, we will find a plan which satisfies a series of *increasing intermediate goals*. To illustrate this, consider the intermediate goal of creating n additional peasants. If we call the planner to find such a plan which satisfies this singular goal, the one produced by MEA will be: produce necessary supply for n peasants; collect enough gold for n peasants; create new peasant n times. However, this plan is not optimal, as the first peasant created can be used to collect resources necessary for creating the other peasants. Even after the scheduler rewrites the plan to allow concurrent actions, the first peasant will not be created before enough gold is collected, and will thus not be able to help produce the necessary supply, which happens before gold is collected. However, we can improve on this by calling the planner to find a plan which satisfies a series of increasing intermediate goals, first of creating an additional peasant, then of creating two additional peasants, etc. The plan produced will now be: produce necessary supply for 1 peasant; collect enough gold for 1 peasant; creating new peasant; repeat whole process n times. Therefore, by using these increasing intermediate goals, we can find plans with potentially shorter makespan.[1]

The pseudocode for our new implementation of the online planner with search is shown along with the pseudocode for the main loop of the planner, in Algorithm 1.

Algorithm 1 Online planner with search: Main Loop. MEA calls the sequential planner, while $Schedule$ calls the heuristic scheduler.

```
1: for every pre-determined number of game cycles do
2:   t ← current time
3:   S ← current game state
4:   if there exists some available actions that can be executed at
     the current time then
5:     Plan ← Schedule(MEA(S, G))
6:     while there exists some unexpanded nodes and search
       time within limit do
7:       G' ← best unexpanded node
8:       {G_1, ..., G_n} ← set of nodes expanded from G'
9:       for all i = 1, ... n do
10:        if G_i is not visited and within search bound then
11:          G_i ← series of increasing goals of G_i
12:          P_0 ← MEA(S, G_i)
13:          S' ← state after executing P_0 from Proj(S)
14:          P_1 ← MEA(S', G)
15:          Plan_i ← Schedule(concatenate(P_0, P_1))
16:          if makespan of Plan_i < makespan of Plan then
17:            Plan ← Plan_i
18:     for all (A_j, t_j^s, t_j^e) ∈ Plan where t_j^s = t do
19:       execute A_j
```

Experimental Results

We now evaluate our new planner in the Wargus domain, by comparing it with our original planner which does not implement the search procedure. The evaluation criteria are time taken to reach the goal, and total planning time. We will also see if the two features we implement, extra search operators and increasing intermediate goals, can help reach the goal faster.

In our Wargus domain, the three renewable resources we need to consider are peasants, which can collect resources and build buildings, barracks, which can create footmen, and townhalls, which can create peasants. Increasing any of them will improve our resource production rate, after we pay a certain price in makespan in creating these additional resources. We will start with an initial state of one peasant and one townhall.

The first experiment we run is to collect 10000 units of gold. The most important factor of achieving this goal is how many peasants we create for collecting gold. It is clear that barracks do not play a part in achieving this goal, and for now, we will also ignore the effects of townhalls on producing peasants. We bound the maximum number of additional peasants in an intermediate goal to be 10, and the time limit for each search to be .1 seconds. The results are showed in Table 1. From the table, we can see that the online planner with search is able to reach the goal slightly faster than without search. As expected, the total planning time needed is now much longer than without search. Moreover, using increasing intermediate goals instead of a singular intermediate goal not only helps the online planner reach the goal faster, but also cut down on the total planning time by more than half.

The second experiment we run is to reach a goal with 20 peasants.[2] Building an additional townhall can be crucial in producing a good plan for creating peasants, because an additional townhall effectively doubles the rate of producing peasants. However, the timing of when the townhall is built is also crucial. If it is built too early, there are too few peasants collecting the necessary resources for the townhall, thereby increasing the makespan, while if it is built too late, this additional townhall becomes less beneficial to reducing the makespan of the plan.

For this experiment, we bound the maximum number of additional peasants and townhalls in an intermediate goal to be 10 and 1 respectively, and the time limit for each search to be .1 seconds. The results are showed in Table 2. From the table, we can see that the online planner with search is competitive with the version without search, but only when

[1] When we have a combination of resources in the intermediate goal, such as townhalls, peasants and barracks, there are many possible permutations of these series of increasing intermediate goals. To cut down on the search time, we can use a pre-determined order of these resources, inferred from the dependency graph of the model, which should give us the best plan. For example, in our Wargus domain, we always first build townhalls, then create peasants (which depends on townhalls), then build barracks (which creates footmen that rely on resources collected by peasants).

[2] For this and the next experiment, we will use a model which needs less resources to build a townhall (600 units of gold, 400 units of wood) and more time to create a peasant (1225 game cycles).

G=10k	Cycles	#P	Plan-time
no search	22500	5	.25s
-O-I	21355	9	26s
+O-I	21620	9	30s
-O+I	**21145**	9	13s
+O+I	**21145**	9	13s

Table 1: Experimental results comparing the number of game cycles, number of peasants created, and the total planning time, to collect 10000 units of gold, for different versions of our online planner. "+O" (or "-O") indicates the presence (or absence) of extra search operators, while "+I" (or "-I") indicates the usage of increasing intermediate goals (or a singular intermediate goal).

increasing intermediate goals are used instead of singular goals.

The final experiment we run is to create 30 footmen. This is the most difficult goal of all three as a plan to build a footmen is more complicated than the other two. Our online planner also needs to consider all three renewable resources. The rate of creating footmen depends on two variables: the number of barracks, and the rate of producing the necessary gold and supply, which depends on the number of peasants available. Moreover, increasing the number of townhalls, while having no direct effect on creating footmen, may have an indirect effect as it can help create more peasants faster. Therefore, there is a delicate balance between the three renewable resources when producing a plan, as increasing one without the other may not necessarily produce a plan with a shorter makespan, but increasing a combination of values may do so.

For this experiment, we bound the maximum number of additional peasants, townhalls and barracks in an intermediate goal to be 10, 1 and 1 respectively, and the time limit for each search to be $.5$ seconds, given the complexity of the plans. The results are showed in Table 3. As expected, when no search is implemented, no additional townhalls are produced, because the online planner never searches for an intermediate goal which creates both additional townhalls and peasants. The best result, which builds an additional townhall and barracks, is found when search is implemented together with extra search operators and increasing intermediate goals.

In general, we find that our online planner is improved by searching over intermediate goals with both extra search operators and increasing intermediate goals implemented. Interestingly, using only extra search operators appears to make our online planner perform worse, likely because the plans produced in this case are far from optimal. However, this is remedied by using increasing intermediate goals in our online planner, which produces better plans. Moreover, total planning time is also cut significantly compared to only using singular intermediate goals, although this is possibly due to the fact that fewer game cycles are needed to reach the goal.

P=20	Cycles	#T	Plan-time
no search	29150	4	10s
-O-I	30040	4	173s
+O-I	33225	4	160s
-O+I	**29090**	3	89s
+O+I	29155	3	89s

Table 2: Experimental results comparing the number of game cycles, number of townhalls created, and the total planning time, to reach a goal of 20 peasants, for different versions of our online planner. "+O" (or "-O") indicates the presence (or absence) of extra search operators, while "+I" (or "-I") indicates the usage of increasing intermediate goals (or a singular intermediate goal).

F=30	Cycles	#P	#T	#B	Plan-time
no search	38830	15	1	1	47s
-O-I	38845	16	1	1	827s
+O-I	42645	22	2	2	1030s
-O+I	36535	16	1	1	591s
+O+I	**33105**	22	2	2	468s

Table 3: Experimental results comparing the number of game cycles, number of peasants, townhalls and barracks created, and the total planning time, to reach a goal of 30 footmen, for different versions of our online planner. "+O" (or "-O") indicates the presence (or absence) of extra search operators, while "+I" (or "-I") the usage of increasing intermediate goals (or a singular intermediate goal).

Future Work

One may expect that such a search procedure over possible intermediate goals would always improve the quality of the resulting plans. However, in our setting, the environment is dynamic and changing. As a result, our action models become more inaccurate over time. For example, in Wargus, as the agent harvests wood, forests disappear from the world map (because they are being chopped down). As a result, the time taken to harvest wood is a dynamically varying quantity. In our models, however, we use a fixed value to represent this duration. Performing extensive search with such inaccurate models may in fact result in suboptimal action selection, because the agent may choose an intermediate goal that decreases the makespan according to the agent's internal model, but increases the makespan when executed in the world. In our future work, we aim to investigate the relationship between model quality, search depth and plan quality for various resource production problems, both empirically and theoretically.

One significant problem of the Wargus game is the implicit spatial properties of the domain. For example, collecting resources and creating new buildings involve travelling of a peasant, and this means that the time taken for the whole action varies as the game progresses, due to the different locations of buildings, obstacles along the way, and the thinning of the forest. Currently, the domain specification, such as the one given in Figure 2, depends on a certain map, and the action durations are given as average values

over game-play in the long run. To remedy the problem, we can implement a learner which adapts the durations of actions while the game progresses, as the client can read in the actual time taken to execute actions and change the domain specification accordingly. The learner is also useful as the client learn the action durations even when they are not given in the beginning of a game. This is particularly useful when we need to achieve the goal in a different map.

As for improving the quality of search, at present many of the parameters, such as the bound on depth and time, and the search operators, are fixed and determined beforehand. These parameters may be varied dynamically throughout the search. For example, at the beginning stage of a game, plans are usually longer, so the time limit should also be longer to allow more nodes should be searched. Moreover, it is also important to determine a close-to-optimal amount of additional renewable resources at that stage of the game. We may also use a different heuristic value other than the makespan of the plan. For example, it may be more beneficial to expand a search node deeper in the search tree.

Finally, at each decision epoch, we use a new search tree to determine the best plan, independent of the search tree at the previous decision epoch. However, these two search trees may very well be closely related, due to the similarity of the two game states. We will investigate real-time search techniques which may help improve the search quality by using the previous search tree as a starting point.

Conclusion

We have extended on our previous approach to solving large resource production problems in RTS games, which works in an online setting by searching over possible intermediate goals that create additional renewable resources at every decision epoch. Instead of using a fixed set of intermediate goals, we use best-first search to generate a variable but bounded set of intermediate goals, guided by the makespan of the plans produced. We evaluate our approach on Wargus and show that it is able to reach large and complex resource goals faster than our previous approach.

References

Chan, H.; Fern, A.; Ray, S.; Wilson, N.; and Ventura, C. 2007. Online planning for resource production in real-time strategy games. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Chen, Y.; Wah, B. W.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research* 26:323–369.

Cushing, W.; Mausam; Kambhampati, S.; and Weld, D. 2007. When is temporal planning really temporal. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 1852–1859.

Do, M. B., and Kambhampati, S. 2003. SAPA: A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20:155–194.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Cost-optimal planning with constraints and preferences in large state spaces. In *International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Preferences and Soft Constraints in Planning*, 38–45.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–203.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predicatable exogenous events. *Journal of Artificial Intelligence Research* 25:187–231.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers.

Newell, A., and Simon, H. A. 1995. GPS: A program that simulates human thought. In Feigenbaum, E. A., and Feldman, J., eds., *Computers and Thought*. Originally published in 1963.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166:194–253.