Online Planning for Resource Production in Real-Time Strategy Games

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. One application of this is the resource production problem in real-time strategy (RTS) games, where players attempt to achieve the goal of producing a certain amount of resources as fast as possible. In this paper, we develop an online planner for resource production in the RTS game of Wargus, where the preconditions and effects of the actions obey many properties that are common across RTS games. Our planner is based on a computationally efficient action-selection mechanism, which at each decision epoch creates a possibly sub-optimal concurrent plan from the current state to the goal and then begins executing the initial set of actions. The plan is formed via a combination of means-ends analysis, scheduling, and a bounded search over sub-goals that are not required for goal achievement but may improve makespan. Experiments in the RTS game of Wargus show that the online planner is highly competitive with a human expert and often performs significantly better than state-of-the-art planning algorithms for this domain.

Introduction

Real-time strategy (RTS) games, such as Warcraft, are games where a player has to engage in actions in real-time, with the objective being to achieve military or territorial superiority over other players or the computer. Central to RTS game-play are two key problem domains, resource production and tactical battles. In resource production, the player has to produce (or gather) various raw materials, buildings, civilian and military units, to improve their economic and military power. In tactical battles, a player uses military units to gain territory and defeat enemy units. A typical 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 this paper, we focus on automated planning in the RTS resource production domain. In particular, the goal is to develop an action selection mechanism that can achieve any reachable resource goal quickly. Such a mechanism would be 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 figuring out how to best achieve it and then manually orchestrating the many low-level actions. In addition to the practical utility of such a mechanism, 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. For example, the planning agent needs to respond quickly to changing goals and inaccuracies in its action models that may emerge as the world map changes.

Most existing planners are not directly applicable to our domain either because they do not handle temporal and numeric domains, or they are simply too inefficient to be useful, or they produce highly sub-optimal plans. A main reason for the inefficiency of existing planners in our domain is that they are intended to be general purpose planners and do not exploit the special structure present in typical RTS resource production tasks. A main contribution of this paper is to expose this special structure and to leverage it for efficient action selection. While doing this precludes generality, the structure is quite universal in RTS resource production and hence our algorithm is immediately applicable to this already widely instantiated sub-class of planning problems.

In our work, we use a simple online architecture that revolves around a key planning component that quickly generates satisficing concurrent plans for achieving any resource goal from any initial state using the minimum amount of resources and minimum number of actions. At each decision epoch, the online planner makes multiple calls to this component, each of which returns a plan that achieves the resource goal through some explicitly selected intermediate goal. The plan with the shortest makespan is preferred, and new actions are chosen to be executed according to this plan, which might be the empty set in cases were the current set of executing actions appears to be best at the current epoch.

The planning component used by our online planner is based on a combination of means-ends analysis (MEA) and scheduling. Given an initial state and resource goal, MEA is used to compute a sequential plan that reaches the goal using the minimum number of actions and resources in the

sense that any valid plan must include all of the actions in the MEA plan. Importantly, the special structure of our domain guarantees that MEA will produce such a plan and do so efficiently (linear time in the plan length). Given the minimum sequential plan, we then reschedule those actions, allowing for concurrency, in an attempt to minimize the makespan. This scheduling step is computationally hard, however, we have found that simple worst-case quadratic time heuristic methods work quite well. Thus both the MEA step and scheduling step are both low-order polynomial operations in the minimum number of actions required to achieve the goal, allowing for real-time efficiency.

While in general planning problems, such interleaving of planning and scheduling might produce no solution or highly suboptimal solutions, we observe that for the resource production domain, our approach is very effective at quickly selecting a good set of actions to begin executing at each decision epoch. In particular, our experimental results, in the RTS game Wargus, show that our planner is able to solve large resource goals in real time. Furthermore the planning performance in terms of the number of game cycles to reach the goal is equal to or better than that of an experienced human player, and is significantly better than the existing general-purpose planners that we were able to apply to our problem.

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, in this paper we 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

Figure 1: A screenshot of Wargus.

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.

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

action collect-gold :duration 300
:require 1 townhall :borrow 1 peasant
:produce 100 gold
action collect-wood :duration 1200
:require 1 townhall :borrow 1 peasant
:produce 100 wood
action build-supply :duration 600
: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 1200
: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.

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.

```
(:durative-action collect-gold
 :parameters ()
 :duration (= ?duration 300)
 :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.

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.

Next, we consider certain properties of our domain specification that 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.

Related Work on Temporal Planning

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.

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. In this section, we briefly describe some of these state-of-the-art planners, and the problems that remain when applying them to our domain.

One issue that is problematic for all planning algorithms is a compact encoding of the domain specification when objects are allowed to be created and destroyed. One way to deal with this aspect is to specify, a priori, a name for each possible object that *could* be created, along with a predicate that tests for its creation. This approach was used to encode the "Settlers" domain in the International Planning Competition (Long & Fox 2003), for example. However, such an encoding is awkward. In particular, at each step, it forces the planner to consider a large number of actions that are impossible. One way to avoid such a cumbersome encoding, which we use in this work, is to use a representation where all units of a given type are treated as exchangeable resources. Thus, instead of having `Peasant1` through `Peasant4`, we may introduce the numeric variables, `total-peasants` and `avail-peasants` that represent the total number and number of currently available peasants. This representation reduces the encoding size and assumes an external procedure to handle the task allocation strategy at a per-unit level. Of course this representation may

not always be appropriate when the planner needs to reason about specific properties of individual units in order to make good decisions. In Wargus, and many other RTS games, the exchangeability assumption along with an appropriate task allocation strategy leads to reasonable performance.

The above representation raises some interesting semantic issues. Since nearly all state-of-the-art planners work with the PDDL specification, the plans they produce respect PDDL semantics. For the resource gathering domain, however, these semantics are somewhat unnatural and restrictive. In particular, the "no-moving-targets" rule of PDDL prevents any pair of actions that use the same resource from starting or stopping at the same time (Kovarsky & Buro 2006), though they are allowed to overlap if their start and stop times are separated by a nonzero constant. Thus, in our exchangeable-unit encoding, actions that use common exchangeable objects now refer to and modify the same numeric variables, and are subject to the no-moving-targets restriction. Observe that, in the encoding where each object is given a "name", this problem does not arise.

While the no-moving-targets restriction is quite unnatural in our domain and can lead to sub-optimal plans, with a small enough "separation constant" it does not in principle prevent concurrency which is critical for achieving small makespans. However, in our tests, no PDDL planner was able to properly exploit the concurrency potential of our domain. In particular, these planners never produced extra renewable resources (e.g. peasants), even when this would have reduced makespan because of the resulting increase in concurrency. Rather the planners would create the minimum amount of renewable resources, typically resulting in no concurrency and poor plans. The exact reasons for these failures require further investigation.

The two issues described above imply that, to our knowledge, no state-of-the-art planner is entirely satisfactory in handling resource production problems. However, several interesting approaches have been developed to handle the other aspects of our domain—durative actions and numeric fluents. 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), all of which we have tested in our work. However, none of these approaches produced satisfactory plans if they produced plans at all. Thus, in our work, we focus on an online heuristic action selection mechanisms that is able to obtain good performance in large RTS resource production problems.

Planning Architecture

In this section, we describe the overall architecture of our planner. In the following sections, we describe the two major components, a sequential planner and a heuristic scheduler, in more detail.

In our work, we focus on an online planning architecture. This seems suitable for the RTS setting where goals and environments change over time. For example, the planner may be used as an assistant to a human that provides high-level goals to achieve as quickly as possible. These goals may change over time, requiring replanning for each change. Similarly, as the game proceeds, the world map and the planner's action models may change, requiring fast replanning with respect to the changed environment. In fact, a changing environment may make it impossible for the agent to execute a plan that is constructed offline ahead of time. Therefore, a planner which takes a long time to plan or does not provide a bound on its planning time is undesirable for our domain even if it may otherwise return the optimal plan. Instead, we aim to develop a planner which finds a good plan quickly while being able to scale with the number of resources, and the number of actions in the plan.

Algorithm 1 Online planner: Main Loop.
The sequential planner MEA is shown in Algorithm 2. The heuristic scheduler $Schedule$ is shown in Algorithm 3.

for every pre-determined number of game cycles **do**
  $t \leftarrow$ current time
  $S \leftarrow$ current game state
  if there exists some available actions that can be executed at the current time **then**
    $Plan \leftarrow Schedule(MEA(S, G))$
    $(G_1, \ldots, G_n) \leftarrow$ a list of intermediate goals which creates additional renewable resources
    for all $i = 1, \ldots n$ **do**
      $P_0 \leftarrow MEA(S, G_i)$
      $S' \leftarrow$ state after executing P_0 from projection of S
      $P_1 \leftarrow MEA(S', G)$
      $Plan_i \leftarrow Schedule(concatenate(P_0, P_1))$
      if makespan of $Plan_i <$ makespan of $Plan$ **then**
        $Plan \leftarrow Plan_i$
    for all $(A_j, t_j^s, t_j^e) \in Plan$ where $t_j^s = t$ **do**
      execute A_j

To adapt to changing goals and environments, 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 means-ends analysis 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.

The rationale for our bounded search procedure over intermediate goals 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 compen-

sated 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, we explicitly find a plan which achieves the intermediate goal of creating some additional fixed amount of renewable resources, such as an additional peasant, then find a plan which achieves 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 this procedure 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. Such solutions can be found by performing more search over these goals and is a topic of future work. In our evaluation, we observed that the heuristic approach of always considering a fixed amount of additional resources as an intermediate goal performed well.

The pseudocode of the main loop of our algorithm is shown in Algorithm 1. Every few game "cycles" (the time unit in Stratagus), the client checks if there are some available actions that can be executed. If so, it calls the planner to find plans which satisfy our goal from the current state. One plan will aim to satisfy the overall resource goal without any intermediate goals, while the others will aim to satisfy each of the intermediate goals which explicitly creates additional renewable resources. The plan with the shortest makespan is preferred, and actions in the plan that should start now are executed. The planner consists of two main components:

- A sequential planner that uses means-ends analysis to find a plan from game state S to goal G with the minimum number of actions, $MEA(S, G)$, and
- A heuristic scheduler which reorders actions in a sequential plan (the output of the previous step) to allow concurrency and decrease its makespan, $Schedule(Plan)$.

Next, we discuss the sequential planner and the heuristic scheduler components.

Sequential Planner

The first component of our online planner is a sequential planner which outputs a sequential plan to achieve the goal from the 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 suffices. Means-ends analysis is a search technique proposed for the General Problem Solver (GPS) (Newell & Simon 1963) and was used in the STRIPS planner (Fikes & Nilsson 1971). It 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 sub-goal. Then from the new state which satisfies the subgoal the process is recursively applied until we reach the goal state. Notice that if one subgoal becomes unsolved while solving another, it will be re-solved later, in the recursive step. The pseudocode is given in Algorithm 2. For simplicity this pseudocode assumes that there is no resource that is both produced and consumed by a single action. It is straightforward to lift this assumption while still maintaining the polynomial-time guarantee outlined below.

Algorithm 2 $MEA(S, G)$.
S, the current game state, and G, the goal, are described in Section 2.

$PS \leftarrow Proj(S)$, the projected game state
if $\forall i$, $R_i \geq g_i$ is satisfied by PS **then**
  return $\emptyset$
$R_i \leftarrow$ some resource where $R_i < g_i$ in PS
$A_i \leftarrow$ action that produces R_i
$r_i \leftarrow$ amount of R_i in PS { Plan to satisfy the R_i goal}
$\alpha \leftarrow$ units of R_i produced by A_i
$k \leftarrow ceil((g_i - r_i)/\alpha)$
$Acts \leftarrow$ sequential plan with A_i repeated k times
$G^* \leftarrow \emptyset${ Plan to satisfy preconditions of A_i}
for all $R_j = p$ which are preconditions of A_i **do**
  if $R_j = p$ is a "require" or "borrow" precondition of A **then**
    $G^* \leftarrow G^* \cup R_j \geq p$
  if $R_j = p$ is a "consume" precondition of A_i **then**
    $G^* \leftarrow G^* \cup R_j \geq k \cdot p$
$Pre \leftarrow MEA(S, G^*)$
$Plan' \leftarrow concatenate(Pre, Acts)$
$S' \leftarrow$ game state after sequentially executing $Plan'$ from PS
return $concatenate(Plan', MEA(S', G))$.

We now informally characterize the behavior of the means-ends analysis in the Wargus domain based on some of the domain properties. First, means-ends analysis repeatedly picks an unsatisfied sub-goal $R_i \geq g_i$, and constructs a sub-plan $Plan'$ which satisfies it. It is possible to show that, given a dependency graph over resources, there exists an ordering over goals such that the MEA procedure will not need to revisit a solved goal. Intuitively, this ordering first solves all renewable resource goals before any non-renewable resource goals (because renewable resource goals, once solved, always stay solved using the monotonic increase property of renewable resources). In this ordering, every action added to the plan is necessary to solve the final goal. Further, if we choose any other ordering of goals that necessitates revisiting previously solved goals, we will only generate permutations of the set of actions produced by the "canonical" ordering. This is because, if we revisit a goal, it must be because some actions used to solve that goal were "used up" by the preconditions of some other goal. Thus, we are effectively permuting the sequence of necessary actions if we choose a different ordering. Since the plan found by MEA has the minimal set of actions, it consumes the minimal set of resources necessary to reach the goal. Finally, because each step of means-ends analysis adds at least one useful action to the plan, its running time is bounded by the minimum number of actions to the goal.

Notice that if the dependency graph between resources contains cycles, it is possible for means-ends analysis to get stuck in an infinite loop for certain initial states. For example, in Wargus, collecting gold requires a townhall and borrows a peasant, while building a townhall or a peasant consumes certain amounts of gold. The presence of these cycles

means there is a possibility that there is no plan to achieve a goal in some cases. However, we can easily extend our algorithm to detect such cases if they happen. Further, as we have noted above, if the initial game state contains a peasant and a townhall, we can guarantee that there is always a plan no matter what the goal state is.

Heuristic Scheduler

The previous component finds a sequential plan. However, to accurately estimate the utility of any renewable resources, we need to reschedule actions to allow concurrency and decrease the makespan of the found plan. 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. The pseudocode for this procedure is given in Algorithm 3.

We can show this procedure is sound using the following informal argument. We need to ensure that when we reschedule an action, every action between the new start time and the old start time remains executable. Now when processing each action, we can always schedule it before a previous action if they do not `consume` or `borrow` the same resource. Consider a pair of actions A and B, A before B, in a valid plan that both `consume` or `borrow` resource R, and assume we are about to reschedule B. First, if A and B are adjacent, the state before A must have enough R for A and B to execute. This means that at that state, A and B could be executed concurrently, or B could be scheduled before A, if possible. On the other hand, if A and B were separated by any actions that produced R in order to satisfy B's precondition, then our procedure would not shift B before the effects of those actions. If A and B are separated by actions not producing R, this effectively reduces to the adjacent case. Thus, this procedure is sound. While this procedure is not guaranteed to produce a plan with the optimal makespan, it is fast (at most quadratic in the number of actions) and performs well in practice. Investigating more complex scheduling algorithms for this problem is a topic of future work.

Algorithm 3 $Schedule(Plan)$.
$Plan = ((A_1, t_1^s, t_1^e), \ldots, (A_k, t_k^s, t_k^e))$ is a sequential plan where the actions are sorted by their starting times in increasing order. The operators $+$ and $-$ add and remove actions from a plan respectively.

```
for i = 1, ..., k do
    t^s ← t_i^s
    R^-(t^s) ← resource state before the effects of actions that end at t^s are added
    while preconditions of A_i are satisfied by R^-(t^s) do
        t^s ← previous decision epoch
    Plan ← Plan − (A_i, t_i^s, t_i^e) + (A_i, t^s, t_i^e − t_i^s + t^s)
```

Empirical Evaluation

We evaluate our algorithm using various resource production scenarios in the Wargus RTS game. In each case, the initial state is set to one peasant, one townhall, and one supply. Every five game cycles, the agent reads the game state from the Wargus engine and calls the procedure in Algorithm 1. If a new plan is found, the agent then sends actions to the game engine for execution. For our experiments, we use a fixed heuristic task allocation algorithm written by a human expert that handles allocating individual units in the game to the tasks assigned by the planner.

To evaluate the performance of our approach, we record the number of game cycles used by our algorithm to achieve the given resource goals. We compare our results with several baselines. First, we report the results achieved by some state-of-the-art planning algorithms on some resource production problems. Next, we compare our results with those of an expert human player in two ways. First, we compare to the cycles taken by the player when playing Stratagus via its player interface, i.e., executing the actions through the graphical interface with mouse and keyboard clicks. We report the best and worst results over five trials. Second, we compare to the cycles taken by executing several plans written by the player, where a series of actions can be specified by one command. For example, he may specify the collection of 1000 units of gold using a maximum of 3 peasants. We also report best and worst results in this case.

Table 1 shows the comparison between the cycles taken by the human player and our planner to achieve some of the easier goals. We observe that, for most large resource goals, our approach successfully creates multiple renewable resources as necessary. Further, our approach is competitive with the human player. In some cases, our planner is able to achieve the goal faster, mainly by finding a better number of peasants to be used to produce the resources. For example, for collecting 1000 units of wood, our planner finds that creating multiple peasants do not help decrease the makespan, while for creating 10 footmen, our planner finds that creating more peasants helps decrease the makespan. Further, we observed that our approach scales well to large resource scenarios. For example, to create 30 footmen, plans typically consist of about 300 actions, and during game-play every plan (including those that create additional renewable resources) is generated within about 0.02 seconds.

In Table 2 we show the results of several state-of-the-art planners on similar problems. We observe that while these planners found valid plans in under a second, we could not get them to create extra renewable resources even when do-

Goal	Human	Human-plan	Planner
G=5k	17865 (5P)	18360 (5P)	**17400 (5P)**
	19890 (3P)	20580 (3P)	
G=10k	**21480 (8P)**	22740 (9P)	22500 (5P)
	24340 (4P)	24150 (10P)	
W=1k	16320 (5P)	16170 (5P)	**14985 (1P)**
	18000 (6P)	18176 (2P)	
W=2k	**19320 (5P)**	19410 (5P)	19980 (8P)
	25350 (2P)	26070 (2P)	
G=5k+W=1k	**20500 (9P)**	21360 (5P)	20585 (5P)
	21440 (5P)	22590 (7P)	
F=5	**20500 (5P)**	21750 (5P)	20515 (4P)
	22300 (6P)	23220 (6P)	
F=10	24640 (7P)	26560 (5P)	**24185 (9P)**
	28950 (9P)	27150 (5P)	

Table 1: Results comparing the cycles taken by a human playing manually (Human), a human-designed plan (Human-plan), and our approach (Planner) to achieve the given goals. The resources are denoted as follows: G–gold, W–wood, F–footmen, P–peasants created during game-play. Best results are shown in bold.

goal	Planner	LPG-td	SGPlan
G=10k	**22500 (5P)**	30000 (1P)	30000 (1P)
W=2k	**19980 (8P)**	30000 (1P)	30000 (1P)
G=10k+W=2k	**28845 (5P)**	60000 (1P)	60000 (1P)
F=10	**24185 (9P)**	45900 (1P)	45900 (1P)

Table 2: Results comparing the cycles taken by our approach (Planner), LPG-td and SGPlan to achieve the given goals.

ing so would greatly decrease the makespan of the resulting plan. We also ran SAPA and MIPS-XXL on these problems, but these were unable to find any plan at all. Finally, TM-LPSAT was unable to scale to these problems because of the large number (over 100) of timepoints required.

Conclusion

We have presented an approach to solving large resource production problems in RTS games. Our approach works in an online setting. Every decision epoch, it searches over possible intermediate goals that create additional renewable resources. For each such goal, it uses means-ends analysis and heuristic rescheduling to generate plans. The best such plan is used to select actions at the current epoch. We evaluate our approach on Wargus and show that it is able to handle large and complex resource goals and it usually finds plans that are comparable to the strategies of an expert human player. In future work, we plan to establish theoretical properties of our approach for various classes of resource production problems. We also plan to study the use of approximate action models in the online planning framework.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *AI Journal* 90:281–300.

Chen, Y.; Wah, B. W.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of AI 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*, 1852–1859.

Do, M. B., and Kambhampati, S. 2003. SAPA: A scalable multi-objective metric temporal planner. *Journal of AI 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. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research* 20:239–290.

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*. Kaufman. Chapter 15.

Kovarsky, A., and Buro, M. 2006. A first look at build-order optimization in real-time strategy games. In *Proceedings of the GameOn Conference*, 18–22.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Newell, A., and Simon, H. 1963. GPS: A program that simulates human thought. *In Computers and Thought*.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166(1-2):194–253.

Wolfman, S. A., and Weld, D. S. 2001. Combining linear programming and satisfiability solving for resource planning. *Knowledge Engineering Review* 16(1):85–99.