Maximization of the Resource Production in RTS Games using Planning and Scheduling

Thiago F. Naves, Carlos R. Lopes
(tfnaves@ufu.br) (crlopes@ufu.br)
Faculty of Computing
Federal University of Uberlândia
Avenida João Naves de Ávila 2121, Uberlândia (MG), Brazil, ZIP 38400-902

***Abstract*—Real time Strategy Games are one of the areas of AI that most present challenges related to planning and constraints satisfaction. In these games, the player needs to produce resources to develop his army in order to attack and defend against enemies. The choice of production goals has a direct impact in the strength of the army. In a previous work, we developed an approach based on Simulated Annealing that achieves quality goals by maximizing resource production. In this paper we extend our previous work by introducing a scheduling component. By intercalating planning and scheduling we were able to find better results than our previous approach. The experiments showed that our system was capable of competing with a human player without any restriction.**

Keywords*—Real-Time Strategy Games, Resources, Goals, Search, Planning.

I. Introduction

Real-time strategy (RTS) games are one of the most popular categories of computer games. "StarCraft 2" is a representative game title of this category. These games have different character classes and resources, which are employed in battles. During the battles, a stage of resources production stands out, where the player must develop the maximum of these in a given interval of time so that he can attack and defend himself from enemies. To produce resources is a complex task since each resource has a series of prerequisites that must be satisfied prior to its production. Thus, maximizing the production of resources during a match is important for victory in the game.

The resources in RTS games are all kinds of raw materials, basic construction, military units and civilization. To obtaining a desired set of resources it is necessary to carry out actions. In general we have three sorts of actions related to resources: actions that produce resources, actions that consume resources and actions that collect resources. A sequence of actions that can be executed in the game is called a plan, and this reaches a certain amount of resources, i.e, a goal within the game. One of the most common problems in maximizing resource production is the choice of goal to be achieved. It is also a problem in most of the studies that involve RTS games. These works focus on finding the necessary actions to achieve a goal of resources, however the goals are determined randomly or without quality assurance for planning.

In a previously work, we developed an approach to solve the problem of choice of goal in RTS games through the maximization of resource production [14]. This make it possible to determine a goal quality along with the actions required to achieve it, in a given time interval in the game. The work was proposed using a sequential architecture in relation to a plan of action and execution of this in the game. In this paper, we present a new version of the choices of goal approach, now using an architecture with scheduling of actions, which improves and increases resource maximization.

The domain used for this work is StarCraft, which is considered the game with the highest number of constraints on planning tasks [5]. In our approach we have to generate an initial plan of action to achieve an arbitrary amount of resources. This initial plan is developed using scheduling in its actions. In order to do this, we have developed a planner system able to plan and schedule the necessary actions to achieve a random amount of resources. The initial plan is the basis for exploring new plans in order to find the one that has largest amount of resources.

In this paper, we focus on the task of finding the goal that maximizes resource production, as we stated earlier. Once we have an initial plan, we use Simulated Annealing [1] (SA) to maximize the amount of resources to be achieved. However, our approach is not only used to specify the goal. We developed a consistency checker capable of handling and validating the changes made in the plan, now also maintaining the scheduling characteristic between the actions of the plan. Thus, our approach when specifying a goal to be achieved develops the plan with all the actions to achieve it within the game.

Our work is motivated by the results obtained in [7] and by gaps in the approaches of [4] and [2] with respect to the choice of goal to be achieved during the production of resources. Thus, this research can go toward a new approach to resource production, and can be extended to other applications. The results obtained are encouraging and show the efficiency of the method to find good solutions. We understand that this research is interesting for the AI planning field because it deals with a number of challenging problems such as concurrent activities and real-time constraints.

The remainder of this paper is organized as follows. In section III related work is presented and discussed. Section II lists the characteristics of the problem. Section IV presents the scheduling architecture developed here, along with the

978-1-4799-0652-9/13/$31.00

two main techniques. Section V discusses the results of the experiments. In section VI we make final considerations about the new version of the approach.

II. Characterization of the Problem

The planning problem in RTS games is to find a sequence of actions that leads the game to a goal state that achieves a certain amount of resources. This process must be efficient. In general, the search for efficiency is related to the time that is spent in the execution of the plan of action (makespan). However, in our approach we seek for actions that increase the amount of resources that raise the army power of a player. This is achieved by introducing changes into a given plan of action in order to increase the resources to be produced, especially the military units, without violating the preconditions between the actions that will be used to build these resources.

To execute any action you must ensure that its predecessor resources are available. The predecessor word can be understood as a precondition to perform an action and the creation of the resource as a effect of its execution. Resources can be labeled in one of the following categories: $Require$, $Borrow$, $Produce$ and $Consume$. Figure 1 shows the precedence relationship among some of the main resources of the domain. These resources are based on StarCraft, which has three different character classes: Zerg, Protoss and Terran. This work focuses on the resources of the class Terran.

```
resource CommandCenter
resource Barracks
resource Scv
resource Firebat
resource Minerals
resource Gas

action build-commandcenter :duration 75 sec.
 :borrow 1 Scv :consume 400 Minerals
 :produce 1 CommandCenter
action build-barracks :duration 50 sec.
 :require 1 CommandCenter :borrow 1 Scv
 :consume 150 Minerals :produce 1 Barracks
action build-scv :duration 13 sec.
 :borrow 1 CommandCenter :consume 50 Minerals 1 Supply
 :produce 1 Scv
action build-firebat :duration 15 sec.
 :require: 1 Academy :borrow 1 Barracks
 :consume 50 Minerals 25 Gas 1 Supply
 :produce 1 Firebat
action collect-minerals :duration 45 sec.
 :require 1 CommandCenter :borrow 1 Scv
 :produce 50 Minerals
action collect-gas :duration 20 sec.
 :require 1 Refinery :borrow 1 Scv
 :produce 25 Gas
```

Fig. 1. Some of the main actions of the resource domain of Strarcraft.

For example, according to Figure 1 to execute an action that produces a $Firebat$ resource you must have a $Barracks$ and $Academy$ available, a certain amount of $Minerals$ and Gas is also required. The $Firebat$ is a military resource. Thus, the action *build-firebat* involves: $Require$ *(1 Academy, 1 Barracks)*, $borrow$ *(1 Barracks)*, $consume$ *(50 Minerals, 25 Gas)* and $produces$ *(1 Firebat)*. The $Barracks$ will be $borrow$, i.e, it is not possible to perform another action until *build-firebat* is finished. The time to build a $Firebat$ is 15 seconds (sec). However, you also need to perform all actions in the shortest time possible, by executing in parallel those that are possible. This situation describes the challenges that surround the planning of actions.

In our approach each plan of action has a time limit, army points, feasible and unfeasible actions. The time limit constrains the maximum value of makespan of the plan of action. The army points are used to evaluate the strength of the plan in relation to its military power. Each resource has a value that defines its ability to fight the opponent. Feasible actions in a plan are those actions that can be carried out, i.e, they have all their preconditions satisfied within the current planning. Unfeasible actions are those that can not be performed in the plan and are waiting for its preconditions to be satisfied at some stage of planning. Unfeasible actions do not consume planning resources and do not contribute to the evaluation of the army points of the plan.

One example of operations in a plan, supposing that a time limit of 170 seconds was imposed to find a goal and the initial plan was set to perform *1 Firebat*. In this case, the completed plan of action has the following actions: *(8 collect-minerals, 1 collect-gas, 1 build-refinery, 1 build-barrack, 1 build-academy, 1 build-firebat)* with 160 makespan and 16 army points. There are several operations that could be made in this plan and that would leave unfeasible actions. For example, exchange one of the actions *collect-minerals* that is a precondition of $Barracks$ by any other action would leave the resource unfeasible. In consequence the resources $Firebat$ and $Academy$ would also be unfeasible, since these have $Barracks$ as one of their predecessors. With this, the plan would have 0 army points.

Some specific combinations of operations can make the plan elevate its resource production and thus its army strength. For example, if it were made the insertion of an *build-marine* action in place of the action collect-gas. The new action will be feasible in the plan, as it will use the minerals left by the action *build-firebat* that became unfeasible due to lack of gas. Now, if in the next two operations the actions of *collect-minerals* and *collect-gas* are inserted in the plan, the resource $Firebat$ comes back to being feasible. This happens because the action of minerals would be made by a resource Scv which is idle and the scheduling process puts its execution time as early as possible in the plan. The gas action would be performed at the same time before leaving the plan, also by the scheduling process. Thus, the final goal now has 28 points without exceeding the time limit, still having the same makespan of 160 sec. The native state of resources when starting a match in StarCraft is *4 Scv, 1 CommandCenter*.

Results, as in the example presented above, are motivators for this research.

III. Related Work

There is little research in maximizing goals of resources production in RTS games. One of the reasons is the complexity involved in searching and managing the state space, which makes it difficult to attend the real-time constraints.

The work developed by [7] remains as our approach in a certain sense. It also uses Simulated Annealing to explore the state space of the StrarCraft. But in this case to balance the different classes in the game by checking the similarity of plans in each class. In our work Simulated Annealing is used

to determine a goal to be achieved that maximizes resources production, given the current state of resources in the game and a time limit for the completion of actions.

[4] developed a linear planner to generate a plan of action given an initial state and a goal for production of resources, which is defined without the use of any explicit criterion. The generated plan is scheduled in order to reduce the makespan. Our work now uses a planner system with scheduling. Our approach deals with dynamic goals for resources production. This is necessary in order to discover a goal that maximizes the strength of the army. To the contrary, [4] works with a goal with fixed and unchangeable resources to be achieved.

Work developed by [2] has the same objective as the one pursued by [4]. These approaches differ in the use of the planning and scheduling algorithms. [2] developed their approach using Partial Order Planning and SLA* for scheduling. [2] also works with a goal with fixed and unchangeable resources to be achieved and can not be adapted to our problem.

The work of [5] presents an approach based on FF algorithm [10] to develop a plan of action that achieves a specific goal in the shortest possible time (minimum makespan). The goals to be used are expert goals, recovered from a database created from analyses of games already played. In this approach, heuristics and efficiency strategies to reduce the search space and achieve better results. In this work the amount of goals and the number of produced resources is limited those goals that are at the database, at some point the game a more appropriate goal cannot be considered by not being at the database or not produce the correct amount of resources. The approach proposed in this paper aims to select and building goals through plans of actions without resource constraints and various possibilities of productions.

The work of [12] and [17] uses an approach with Case Based Reasoning (CBR) and Goal Driven Autonomy (GDA). CBR is used to select expert goals from a database and GDA allows a goal to be discarded during its executionand a new goal is chosen that takes into account the game situation. This approach has similarities with that proposed in this article, especially in relation to planning goals within the game, however this approach also has a limit on the number of goals that are considered, which are present in a database.

We also surveyed some already existing planners that could be applied. Approaches described in [6], [9] and [11] were considered for our problem at hand, but both have a different approach and would have to be modified to adapt to our goal. In short, most of the approaches surveyed have different objectives and the techniques used are not efficient for the domain that we are exploring.

IV. Scheduling Architecture

This architecture has been developed and adapted so that the choice of goal approach could operate with the actions executed in parallel, which increases the ability to maximize resources and get even better results. The results are very similar to those which are obtained by experienced players of RTS games, in particular StarCraft. Experienced players tend to parallelize most actions executed during a match.

As in the sequential architecture, there are two techniques that integrate the SA allowing the maximization of the amount of resources, which are the system planner and scheduled consistency checker. The parameters of the SA and its algorithm and way of operating were not altered when using the architecture with scheduling. Thus, the relevant details to SA will not be discussed here and emphasis will be given to new architecture. The details about the operation of the SA in choice of goal approach are detailed in [15].

A. Planner System

Planner system uses the concept of partial order planning (POP) [13] to generate a plan of action and perform the scheduling task during this process. Its development was supported by need of a planner that can generate a plan of action as input to the SA, enabling the search for solutions in with parallelism of actions. Another need was to find an integrated solution that reduces the time spent during the process of planning and scheduling.

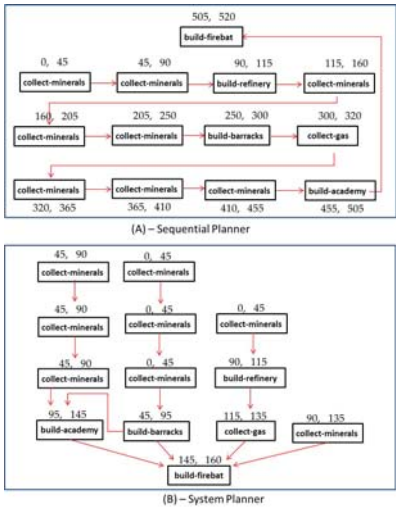

Fig. 2. Comparison between the plans obtained using planner system and sequential planner.

The main motivation for constructing the planner is in the significant gain that the production resources have in relation to the sequential planning, since the choice of a quality goal occurs by maximizing this production. Figure 2 illustrates this advantage in the production of resources. In this figure, the plan (A) illustrates the actions required to build the $Firebat$, where these had execution times assigned and are executed in the linear order indicated by the arrows. This plan takes 520 seconds to perform the actions. In the plane (B), the planner determines the actions necessary to build resources and schedule them. In this plan, the arrows represent the causal links between actions. These indicate the restraining order and resources that will be produced and used between actions. This order specifies only that an action should be executed before another action of which it is a precondition, but how much earlier is indifferent. Thus, the order of execution of actions does not follow a fixed ordering as the sequential planner. Figure 3 shows the execution order of the actions.

The architecture of the planner is composed of two levels. In the first, the processes of the constraint satisfaction problem

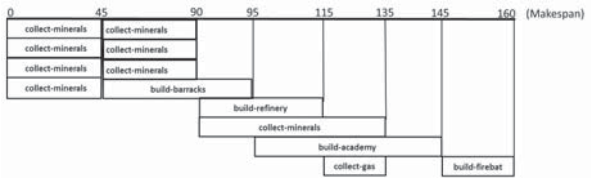

Fig. 3. Order of execution of the plan of action generated in Figure 2.

for the composition of the plan are made, in the second the actions defined by this process are subjected to scheduling. These two levels intercalate its executions and are described in Figure 4. With this architecture, the algorithm has the concept of strong coupling [8] where planning and scheduling problems are reduced to a uniform representation. If any inconsistency is found in the scheduling step it is possible check it out and satisfy it in the planning stage being performed intercalated with that. Thus, it is not necessary to interrupt the process for such verification. In this coupling, the planner is essential for successfully achieving this representation, because the causal links defined between the actions in the planning stage have the temporal order of constraints among the actions, which also assists in the production and use of resources. With that, the scheduling step only finds the best time for the action to be performed, leaving planning tasks where resources information and constraints between actions are checked and set, to be used in the next stage.

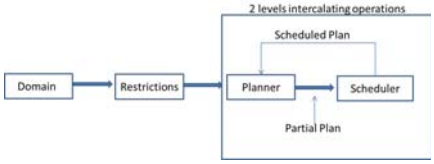

Fig. 4. Representation of the strong coupling intercalating POP and scheduling.

POP allows the coupling in a more intrinsic level to be reached, if used, for example, the sequential planner such representation would be decomposed into two separate sequential problems. Finding a sequential plan of action and since it has been finished would start the scheduling process, similarly to the works of [4] [2]. Algorithm 1 describes the pseudo-code of the planner system.

The planner receives as parameters the list where the actions that will make the plan will be inserted $Plan$, the resource that is the current goal R_{goal} and the time limit for completion of the plan of action T_{limit}. A initial plan of action that achieves an amount of resources in an interval, for example, 180 sec can produce many more resources than the sequential planner. At the beginning of the operation, the planner calls the method $LinkBuild()$ (line 1), responsible for defining all the actions that will make the plan and its respective causal links. This method then sets the links of partial order planning, inserts the first action in the plan from the goal resource and from its predecessor resources fills the plane with the remaining actions necessary.

The algorithm between lines 2 and 5 verifies if the action is not the type of production unit, i.e, whether the action is

Algorithm 1 Planner($Plan$, R_{goal}, T_{limit}))

```
1: Plan ← LinkBuild(Plan, R_goal)
2: for each Action Act ∈ Plan do
3:     if Act.unity = false then
4:         times ← Quantity(Act)
5:     end if
6:     while i < times do
7:         for each Action ActRs ∈ Plan do
8:             if ActRs = Act.rbase then
9:                 Schedule(Plan, ActRs, Act)
10:            end if
11:        end for
12:        contr ← Constructs(Plan, Act)
13:        if contr = false then
14:            Exit()
15:        end if
16:        i ← i + 1
17:    end while
18: end for
19: return Plan
```

the mineral or gas type, but only resources that are not of type unit. If so, the function $Quantity()$ (line 5) is called to check how many times the action should be executed. For example, if the action *build-minerals* has a link to a *build-academy* that requires 150 $Minerals$, three executions of action are necessary because each execution generates 50 $Minerals$.

All resources of the game have a resource base that is responsible for building the resource within the game. For example, the Scv is base resource of $Barracks$, once it is the Scv that builds the $Barracks$ in the game. Whenever the action corresponding to a resource base is found, the method $Schedule()$ (line 9) is used. It traverses the list that contains actions executed by the resource base and tries to find a time interval between these so that the current action can be executed. The scheduler method uses a strategy to find the shortest execution time between the base resources that are in the plan, and can also execute the action.

After setting the starting time of the action through the scheduling process, the planner will now finish setting its attributes, again using the planning stage. The method $Constructs()$ (line 12) is responsible for this task. This sets the starting and ending execution time of the action, inserting new actions of renewable resources if necessary, modifying some link action if necessary due to the scheduling that can change these links, also confers there is not any threat in these (according to the concept of POP), and checks to see if the time limit has not been exceeded in the plan. The final plan that is built by the planner, contains the actions that reach a random amount of resources, as well as having all actions.

B. Scheduled Consistency Checker

With the search for goals toward plans of actions that have quality in army points and yet scheduled actions, verification and validation of new plans becomes even more important. The scheduled consistency checker developed is used in this step. Algorithm 2 shows its pseudo-code.

The algorithm 2,receives as parameters the current solution of the SA $Plan$, the operation that will be performed in the plan opt, the time limit of the solution will be generated T_{limit} and finally the variable $penalty$ for its value be defined. Initially the method $Invib()$ is called (line 1) responsible for

Algorithm 2 Consistency($Plan$, opt, T_{limit}, $penalty$)

```
1: Invib(ActsInv, opt, penalty)
2: for each Action Act ∈ ActsInv do
3:    if Act.feasi = true then
4:       for each Action Actl ∈ Act.link do
5:          UndoL(Actl, ActsInv, penalty)
6:       end for
7:       for each Action Actl ∈ Act.Clink do
8:          ReleaL(Actl, ActsInv, penalty)
9:       end for
10:   end if
11: end for
12: Rearrange(Plan, ActsInv, opt)
13: while cont = true do
14:    for each Action Act ∈ ActsInv do
15:       feasible ← Gather(Plan, Act, penalty)
16:       if feasible = true then
17:          for each Action ActRs ∈ Plan do
18:             if ActRs = Act.rBase then
19:                Schedule(Plan, ActRs, Act)
20:                cont ← Constructs(Plan, Act)
21:             end if
22:          end for
23:       end if
24:    end for
25: end while
26: return Plan
```

checking what actions will be unfeasible due to operation. The method places the first actions in the unfeasible action list $ActsInv$, which are necessary for the algorithm to find out which others will also become unfeasible.

The algorithm between lines 2 and 11 finds the other actions that will be unfeasible. For each one that still has the attribute $feasi$ valid (line 3), i.e, every action that has just been added to the list of unfeasible, the algorithm calls the methods responsible for removing its links and finding out which other actions will also be unfeasible. The method $UndoL()$ (line 5) is called to put each resource that receives a link of action that was unfeasible Act within the list of actions unfeasible. This is done as Act which is a precondition of such actions, which will no longer run. For example, if an action *build-refinery* become unfeasible this means that the *Refinery* does not exist in the game anymore, so all actions *collect-gas* receiving a link of this resource will also be also unfeasible.

The method $ReleaL()$ (line 8) is called just after $UndoL()$, it is responsible for releasing the actions that were serving as precondition for the action that became unfeasible, thus eliminating the links from other actions that came to it. This frees the resources used by the action to be used by other actions. After the list of actions unfeasible has been completely filled, the algorithm calls the $Rearrange()$ (line 12). It performs the operation on the plan so that it becomes a new neighbour plan. From the line 12 onwards, the algorithm focuses on what actions can becomes feasible again, configures its attributes and schedules it. The function $Gather$ (line 15) is used with every action unfeasible. It seeks the preconditions of the action that is being checked Act, to see if the available actions can be used to execute it.

When the preconditions of an action are found within the plan, the algorithm schedule its and sets its attributes, as they will become feasible. It made a search for base resources of the action (line 17) to define which among those available would execute the action. When a base resource is found the scheduling algorithm is called $Schedule()$ (line 19). The algorithm is the same as used in the planner. It performs the schedule for each action returns to be feasible.

Finally, the function $Constructs()$ (line 20) is called to set the attributes of the action that returned to be feasible. It sets up the execution times of actions, builds all the links between them and their preconditions, removes the action from the list of unfeasible and checks to see if the time limit has not been reached. If this time is reached, the action does not return to feasible and the verifier stops the process returning the plan with the last action that became feasible as new solution for SA.

V. Experiments and Discussion of Results

The experiments were conducted on a computer intel core i7 1.6 GHz CPU with 4 GB of ram running on a windows operating system. The API Bwapi [3] allows the control of StarCraft units and and information retrieval such as the number of cycles and time of the game. It was used as an interface for the experiments with a human player and for evaluating the SA performance.

The procedures used in the experiments are: $SA(S)$, $Player(A)$ and LT. $SA(S)$ refers to our approach, with SA plus planner system and scheduled consistency checker. $Player(A)$ is a human player experience level in the game. To be classified as an experienced player a 5-year experience with StarCraft was considered. LT is a A.I agent player (bot) called LetaBoot [16], who participated in the 2014 AIIDE StarCraft Competition [1], getting in the top three of the competition.

In all tests, the used approaches tried to achieve the best goal with resource production, i.e, a goal focused on produce resources that allow a player to advance against the enemy bases and overcome it in a confrontation. These goals produce various types of resources that increase the amount of army points. Producing resources with ability to fight enemy leads to the production of other types of resources present in the game, since its are required to enable more powerful resources. In fact, a feature observed in various players when played matches of the game were analysed is the trend to produce offensive resources along the match and only produce other types of resources when these are needed to enable the coming of new resources that will be used to attack and resist to the enemy's army. Thus, the approaches will be compared according the amount of resources that can produce within a time limit.

TABLE I. Results of Experiment 1

Time Limit	Player(A)	SA(S)	*Makespan* of SA(S)	Runtime of SA(S)
150 sec.	29	**32**	150 sec.	26,1 sec.
300 sec.	140	**141**	299 sec.	48,2 sec.
450 sec.	**206**	182	450 sec.	98,02 sec.
600 sec.	**306**	245	598 sec.	141,1 sec.
750 sec.	**437**	302	748 sec.	197,0 sec.
900 sec.	**601**	400	899 sec.	254,9 sec.
1050 sec.	**781**	511	1047 sec.	301,6 sec.

[1]StarCraft Competition is a competition between Starcraft bots, this competition is an event that is part of AIIDE (AAAI Conference on Artificial Intelligence and Interactive Digital Entertainmen). Available at http://webdocs.cs.ualberta.ca/ cdavid/starcraftaicomp/.

In Experiment I, we compared the performance of experienced player with SA(S). The army points (below each procedure in table) obtained in this experiment are much higher than those obtained in experiments made with the sequential architecture. This stresses the importance of scheduling in this approach. The algorithm was able to overcome the player on three occasions and was overrun by another four. It is interesting to note that the algorithm achieved the best results in the mean time intervals (up to 400 sec). In these results the $SA(S)$ runtime was low enough so that the approach was tested inside the game. This is due to the causal link structure of the POP approach, which helps reducing the search for actions and preconditions during management. However, with the growth of time limit intervals, the runtime of SA(S) increases dramatically. This is due to the rise in the number of actions, which begins in the generation of the initial solution by the planner. When solutions are generated by $SA(S)$, the complexity of validating these increases along with the number of actions that are present.

In order to use our approach under real-time constraints we used a strategy that takes into account the fact the best results have been achieved considering time intervals that range from 150 to 300 seconds. This strategy consists of decomposing a resource production goal to be achieved in time intervals superior to 300 sec into smaller resource production subgoals to be achieved in time intervals in which the algorithm can find the best results. As soon as a subgoal is achieved its plan of actions is carried out. While executing the actions that satisfies a subgoal, the algorithm keeps working in order to figure out a plan of action that satisfies the next subgoal, and so on. For example, suppose a 600 sec time limit for resource production. In this case, the algorithm splits the resource production goal for this time interval into three resource production subgoals considering a 200 sec time limit for each one. Whenever a goal is divided into subgoals, these have time intervals between 150 and 200 seconds.

TABLE II. RESULTS OF EXPERIMENT 2

SA		**LT**	
Runtime	Army Points	Army Points	**Execution**
41,1 sec.	**112**	94	**1**
43,0 sec.	92	**101**	**2**
41,9 sec.	**106**	93	**3**
44,3 sec.	**109**	94	**4**
42,8 sec.	**98**	90	**5**
44,2 sec.	**112**	94	**6**
41,6 sec.	94	**101**	**7**
41,1 sec.	**106**	39	**8**
43,5 sec.	**104**	90	**9**
44,9 sec.	**102**	94	**10**

Experiment 2 compares the resource production of SA and the bot LT. LT bot was chosen because it is a bot with open code and also by the use of the resources of Terran class in Starcraft, the same used by our approach. To accomplish this experiment tests were performed to determine the maximum time that LT was able to produce resources without using their resources on direct attacks within the game. This was considered the time that both approaches would have to perform their resource production. At the end, the quality of both in terms of army points was evaluated. LT also was chosen for having an offensive production resources strategy, similar to that described at the beginning of this section. During each experiments execution the bot concentrated only on achieving goals that increase its ability to confront and be confronted by enemies. The time was set in 270 seconds and ten executions were carried out with this time.

In the table II, the SA overcame LT in eight executions and lost two others in relation to resource production. The LT has achieved 101 army points in three executions when 3 Wraith (Terran resource) was produced. In other runs, the agent showed a pattern of army points that varies between 90 and 94 points. The military resource production of LT follows a pattern where the resource Wraith is the most produced, already SA seeks to produce resources that add up more army points to the goal being determined, without following a pattern of production. The LT strategy has a maximum amount of resources that can be built in a time interval and thus the values of army points achieved by it were often similar. $SA(S)$ does a search for the best combination of resources that maximize production and raise the power of the army produced.

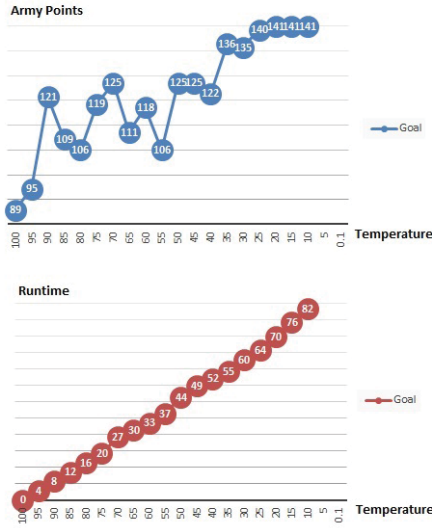

Fig. 5. Graphs of army points and runtime for one of the runs of the experiment I.

In Figure 5 we depict the graphs for the army points and runtime with time interval of 600 sec in Experiment 1. In relation to the army points, $SA(S)$ oscillates between solutions at low temperatures, so that in some cases it even considers plan of action with lower value to the initial plan. This is due to the high amount of changes that occur in the plan when made an operation, even a simple operation. At low temperatures the algorithm follows the normal trend of accepting mostly better solutions than the current. In relation to runtime, the algorithm maintains the tendency to work more quickly when the temperature is in medium/high values. At lower temperatures it spends more time.

The Experiment III, shows the performance analysis of the $SA(S)$. In this, we also made 15 runs of the algorithm for three different time intervals. The algorithm has its performance evaluated in average performances that are close to the optimum value. Good results were obtained with the average intervals. With a limit of 450 sec we obtained solutions almost

TABLE III. RESULTS OF PERFORMANCE TESTS OF SA(S)

Time Limit	200 sec.	450 sec.	750 sec.
Optimum value for army points	55	209	442
Number of runs over 95% of optimum	82%	37%	8%
Number of runs over 90% of optimum	12%	25%	28%
Average value	77	179	301
Average CPU runtime	27,5 sec.	97,3 sec.	199,7 sec.
Number of Runs	15	15	15

exclusively to 90% and 95% of the optimum. With intervals of 750 good results were also achieved. With 1050 sec the algorithm had more difficulty maintaining the results as in the previous experiment, due to the increase number of actions.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented a new approach for improving the maximizing the production of resources in a RTS game. We extend our previous work by developing a system capable of intercalating planning and scheduling of actions. An important component in our system is a consistency checker that enables the generation of new plans with parallel actions.

The objective of this new architecture was to achieve satisfactory results regarding the quality of the solutions, as we achieved in a previous work. With this new approach the results were even better, since the approach increased the maximization of resources while competing with human players without any restriction and maintaining runtime compatible with the game.

As future work, we aim to implement different scheduling algorithms in order to improve the system performance. Also, we intend to integrate our algorithm for resource production with algorithms that implement strategies for attack and defence to build a complete AI player.

ACKNOWLEDGMENTS

This research is supported in part by Research Foundation of the State of Minas Gerais (FAPEMIG) and Faculty of Computing (FACOM) from Federal University of Uberlândia.

REFERENCES

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimisation and Neural Computing.* John Wiley e Sons, Inc. New York, NY, USA 1989, 1989.

[2] A. Branquinho, C. R. Lopes, and T. F. Naves, "Using search and learning for production of resources in rts games," *The 2011 International Conference on Artificial Intelligence*, 2011.

[3] Bwapi, "Bwapi - an api for interacting with starcraft : Broodwar," available in http://code.google.com/p/bwapi/ - Last visited on 09/04/2014.

[4] H. Chan, A. Fern, S. Ray, N. Wilson, and C. Ventura, "Online planning for resource production in real-time strategy games," in *ICAPS*, 2007.

[5] D. Churchil and M. Buro, "Build order optimization in starcraft," *AIIDE 2011: AI and Interactive Digital Entertainment Conference*, 2011.

[6] M. B. Do and K. S., "Sapa: A scalable multi-objective metric temporal planner," *Journal of AI Research*, vol. 20, pp. 63–75, 2003.

[7] T. Fayard, "Using a planner to balance real time strategy video game," *Workshop on Planning in Games, ICAPS 2007*, 2007.

[8] E. Fink, *Changes of Problem Representation: Theory and Experiments*. Springer, 2003.

[9] A. Gerevini, A.; Saetti and I. Serina, "Planning through stochastic local search and temporal action graphs in lpg," *Journal of Artificial Intelligence Research 20:239-230*, 2003.

[10] J. Hoffmann and B. Nebel, "Ff: The fast-forward planning system," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[11] S. Kecici and S. S. Talay, "Tlplan-c: An extended temporal planner for modeling continuous change," *The International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.

[12] M. Klenk, M. Molineaux, and D. Aha, "Goal-driven autonomy for responding to unexpected events in strategy simulations," *Computational Intelligence*, pp. 187 – 203, 2013.

[13] S. Minton, J. Bresina, and M. Drummond, "Total-order and partial-order planning: A comparative analysis," *Journal of Artificial Intelligence Research*, vol. 2, pp. 227–262, 1994.

[14] T. F. Naves and C. R. Lopes, "Maximization of the resource production in rts games through stochastic search and planning," *IEEE international conference on systems, man and cybernetics - SMC*, pp. 2241– 2246, 2012.

[15] ——, "Stochastic search and planning for maximization of resource production in rts games," *International Conference on Artificial Intelligence - ICAI*, pp. 2241– 2246, 2012.

[16] S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M.Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.

[17] B. G. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game ai," *In Proceedings of IEEE CIG*, 2010.