WILL SCHEDULERS BE AVAILABLE ON BOARD IN THE NEXT GENERATION OF ROBOTS ?

Thierry Fayard

Centre National d'Etudes Spatiales
18 av Ed. Belin
thierry.fayard@cnes.fr

Abstract

This paper presents a simulated annealing solution procedure for a resource constraint project scheduling problem (RCPSP) as it may be found in future space robotic applications. Such RCPSPs are also found in some game problem where it is easy to compare computerized solution with human play. The player, or the computer, has to find the best sequence of activities in which an objective function is maximized within a given time. The scheduling problem found both in robotics application and in such a game, is more difficult than usual because it is subject to temporal constraints, and to constraints for both renewable, non renewable and cumulative resources. The cumulative resources are depleted or replenished over time depending on the player's choice. A simulated annealing algorithm is successfully tested, especially the model efficiency, the neighborhood definition, which is crucial, and the choice of an objective function which may represent complex aims. It is shown that the performances can satisfy a good player and then such tools may be of some help in robotic applications. It is also shown that it is robust because the heuristic is always able to find rather good solution. And it is realistic since the computing power needed will be available on board in a medium-term future.

Keywords : simulated annealing, heuristic, project scheduling, resource constraint, robotic.

1. Introduction

This paper describes a simulated annealing (SA) procedure to solve the resource constrained project scheduling problem (RCPSP), as it may be found in future space robotic application. Standard RCPSPs is a problem where you have a number of activities to execute under some time or resources constraints and you must achieve a feasible sequence able to minimize the total makespan of the project. It is a well known and well surveyed problem and many algorithms have been studied, but, for real-life problems only meta-heuristic approaches are able to give good solutions when the size is over 50-60 activities. A typical RCPSP is minimizing the production time in a factory, where workers and machines are the resources and you must proceed some activities to achieve a production. The problem we consider is slightly different since we don't need to minimize a production time, but we need to optimize an objective function within a given duration. For example we have some equipments on a space vehicle and we want to conduct as many experiments as possible and analyzes during the martian day. There is an other difference with classical RCPSP ; to solve realistic situation in space robotics, we need to consider renewable, non renewable and cumulative resources, the latter has been studied only recently (Neumann, Schwind 2002). They said in conclusion of their paper "an important area in future research is the discussion and efficient solution of RCPSP subject to temporal constraints for both renewable and cumulative resources". Actually, the real world problem with cumulative resources, and with complex time and precedence constraints is a totally new domain of investigation. The question asked in this paper is : "will scheduling technology be of some help in the mid-term future for space robots ?", similar to the title question. We will try to show that the answer is probably yes. To be affirmative we need to demonstrate 3 points : it will be useful, it will be safe and, last but not least, it will be possible considering hardware technology available in space at that time.

Actually, in space industry scheduling technology are not commonly used, both on ground system or on board, particularly scheduler with meta-heuristics such as Simulated Annealing (SA) or Genetic Algorithm (GA). At CNES, a scheduler was developed for the Rosetta operations, to find a feasible schedule under time and precedence constraints, without any optimising capabilities. This software is ground based. Another ground application developed under the responsibility of the CNES is the SPOT picture snapshot planer. The goal of this tool is to optimise the client's satisfaction under some temporal and resource constraints. At CNES as well, some ideas have been put forward about a

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **13 JUL 2005**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **šWill Schedulers Be Available On Board In The Next Generation Of Robots ?**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Centre National dEtudes Spatiales 18 av Ed. Belin**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001791, Potentially Disruptive Technologies and Their Impact in Space Programs Held in Marseille, France on 4-6 July 2005. , The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**10**	

scheduler for the robot generation presently under studies, but the decision trees were small enough to be totally explored, thus rendering more sophisticated heuristics useless. We can also think that meta-heuristic technologies are random based, and the need for proved reliability in space is not in favour of such an approach.

So the present state doesn't make us enthusiastic about the use of meta-heuristic scheduler for space applications, especially on board. But this may change for 2 reasons. First in the medium-term future, (10-20 years ?), distant robot mission may need on board software to optimise locally taken decisions, mainly because missions are more complex than before so that the choice combinatory will increase significantly. By distant mission, I mean Mars orbit and beyond, so the reactivity of a humanly taken decision can't be better than 40 minutes, meanwhile some dozen decisions can be taken on board. Second, meta-heuristic scheduling technologies come to a point where they are able to deal with more complex situation than before, and not only for academic cases, but also for real-life problems. For these reasons combined, I suspect that some future missions, with a strong demand for autonomy, will have a different design in term of operation schedule and will implement on board decision capabilities, where meta-heuristic schedulers can operate.

The problem now is to be sure that such schedulers are able to do good work. A formal proof of efficiency for such a scheduler is not available yet, and it will be so for quite a long time because of complexity. The idea comes to test it against humans, but to be convinced we need a representative base of test. Actually, I already tried a simulated annealing scheduler to act as a player in a very well spread class of strategy game known as “Real Time Strategy” or RTS game. If the scheduler is able to do as well as world class players among a community of millions of people who has played a thousand millions time, I presume it is a good scheduler. And it is the case, a paper is under work to present these results in a more formal way than here. We will see how a wining scheduler can be used in a robot mission, in a safe way, and with available on board hardware technologies, the 3 points needed to answer the title question, as mention before.

This paper is organized as follows. First we present a model for a hypothetical robot mission with a RCPSP to be solved in Section 2, in Section 3, we transpose that RCPSP into a problem found in some game situation and we show that it is similar to the previous one. We discuss implementing simulated annealing methods to optimize the objective function of such scheduler in Section 4. Then we analyze some results, we show that the scheduler is efficient compared to human choices and we evaluate the reliability of the scheduler and its computer need in section 5. Finally we discuss some perspectives.

2. A model for a hypothetical space mission

2.1. A academic case robot

Let's define a robot, or a vehicle with some scientific equipments inside as a pure academic case. The purpose of this vehicle can be : find some evidence for life, or understand the geology of the site. It will have to collect various samples on the planet surface, to produce some analysis, to take some pictures, to move, and to send data to Earth. The robot needs energy which will be provided by solar panels. Energy will be stored in a battery. Some activities will produce data to be stored in memory before sending to Earth when it is possible. We assume that the robot may encounter yellow, red and black samples. The robot is able to recognize them and to move toward them.

2.2. The general planning problem with such robots

The activities can be listed in short in this way:

- Move and take picture
- Collect sample
- Do an experiment, it can be : smash a stone, heat it, do chemical or physical measurement…
- Send data to Earth

When the robot is left alone it will first take some pictures of the environment, then it will evaluate the distance to each site where samples are available. Then the onboard computer will run the scheduler to make a plan of activities. The plan will look like : move to a site, collect sample, do experiments, move to another site, maybe still while doing experiments, send data to Earth, collect samples, etc.

Moving is power demanding but it will be necessary for finding different type of sample. Most experiments will use energy and fill memory with data. If some steps always need to be done before another one, they can be clustered to simplify the schedule problem, for example it may always need to smash a stone before some analysis. Depending on the number of experimental units and the total possible experiments and their dependencies the schedule problem can be relatively tough. This problem is much more complicated than usual RCPSP due to the resource reservoirs, and also because the execution mode of activities may change along the plan.

2.3. The resources

We will define 3 types of resources in a way which is convenient for this problem.

The renewable resource : most scientific equipments are renewable resource since they are able to start again an activity when the previous one is finished.

The non renewable resource : a sample can be considered as a non renewable resource since once it is used it cannot be used a second time. The same for a chemical product dose which may come in use for some experiments. But also an equipment which can be used only let’s say 10 times, can be considered as 10 non renewable resources.

The resource reservoirs (also called cumulative resources) : the on board memory can be considered as a resource reservoir which will store the data generated by the experiments. The battery is also a resource reservoir to store energy. Both are replenished or emptied over time by activities under process.

2.4. Time constraint

The rule is that, to perform an activity, you have to wait for the resources to be available.

An activity is thus feasible when :

- Needed renewable resources are ready, or will be ready when some previous activities come to an end.
- Needed non renewable resource are available.
- Needed energy or memory are available in resource reservoir, or at least one unit is able to collect them.

When an activity is feasible, it may take some time before it actually starts. Maybe because energy has to be collected with solar panels or memory has to be freed by sending data to Earth, or because a needed experimental system is already doing something which has to be terminated first. Waiting for resources to be available introduces lags. This decreases the efficiency of the project ; a good schedule will thus try to minimize delays.

Also some temporal constraints can be very strong, such as sending data to Earth, since the visibility with an orbiter or the earth has to be guaranteed.

2.5. The scientist’s goal

Of course the scientific result cannot be evaluated by a simple function, but we assume that the scientist is able to quantify the importance of the data in advance by giving a weight to each experimental result. The weight may evolve to refine the scientific strategy, but we won’t consider here any interaction with the environment or the scientist desire, we just want to schedule a plan which is able to optimize a function. That one has to be as near as possible to the scientist’s needs. The weight of an individual result may depend of the occurrence of such a result. For example suppose that the scientist gives a strong weight to the spectral analysis of smashed black stone, and a weak weight to red stone analysis, the scheduler may process only black stone, avoiding red one. But the scientist may need at least little red stone analysis, to overcome such problem the scheduler should be able to modify weight according to what has already been done.

2.6. How to test the robot scheduler

A scheduler has been computed, but since there is no simple way to evaluate its performances against a human, the idea is to find a model which is similar and where solutions were found by a large community of human actors. I decided to compute the scheduler for a real time strategy game (RTS), where we find all three types of resources, and time constraints. The problem must also optimize a function for a given duration, which is perfectly relevant in RTS game situation.

3. A model for real-time strategy game

3.1. What is a Real-Time Strategy Game ?

In most real-time strategy games, competition based on resource gathering and killing enemies takes place on a two-dimensional map. On the map there are units, buildings and some collectable materials. Units and buildings may belong either to you or to your enemies, and you can control only your own. Most materials on the map are available for all, in fixed quantities or infinite, as long as you are able to collect them. They can be wood, food or mine, depending on the context of the game. A unit is an entity able to move on the map ; it can be a soldier, a tank, an aircraft, a resource collector or any moving device. Units are mostly fighters : they are able to destroy other units or buildings. They can be specialised in ground attack (short- or long-range), air attack or resource gathering. The buildings are used to produce units, upgrade units, allow materials storage, or develop specific capabilities which enable production of some kind. Materials are required to produce a unit, a building or an upgrade, and they must be available at the time you decide to produce it. They have to be collected constantly by the players. There are many options to win the game, but to destroy all enemy units and buildings is the most common. If you play against a human, you need an army at least about the same size as his to achieve victory, and in addition, you also need to constantly produce new units to replace lost ones.

In short, we can distinguish two phases in such a game : a ‘development phase’ and a ‘fighting phase’. Each player starts the game with some units and/or buildings, then he develops his army ; this is the ‘development phase’. When he feels strong enough he starts to move his fighter units to engage the enemy, this is the ‘fighting phase’. Although these two phases can be partly simultaneous, we only consider the ‘development phase’ as an independent whole in this paper. In practice, this limitation is of no consequence for the player for two reasons. First, most of the time there is no interaction at all between players in the beginning because they start on distant places on the map. The time taken by a player to move his army from one place to another will be used by his the adversary to produce a much stronger army, and this effect is particularly relevant at the beginning. Second, once the player has chosen a tactic, the set of activities to be schedule doesn’t depend on the decision taken by his adversary. The player has only to guess the most appropriate tactic with his limited knowledge of the opponent’s intention. In brief we will, here, only consider a schedule problem and omit the multi-agent theories.

3.2. The general planning problem in RTS

During the ‘development phase’, playing means executing one of the four type of activities and nothing else:

- sending a collector unit to materials
- building something
- producing units
- upgrading units.

Collecting materials is an automatic process which usually doesn’t need the player’s attention, as for other types of activities. Building something can start as soon as the required resources and the materials are available. The same applies for producing units and upgrades. It ends after a given time which may differ also due to the state of the game. For example, some upgrades may reduce the duration of building. In some games, one can use more than one unit to speed up the building process.

Any activity can be performed many times while resources are available and the dependence-feasible graph is respected. After the start, activities can run concurrently. Activities can be performed in alternative modes which differ with regard to processing time or resource requirements. The problem to be solved is thus a multi-mode resource-constraint project scheduling problem (MRCPSP). In the present context, a project is a play, or a sequence of activities (SOA) ordered by their starting dates. Then the scheduler has to find the best SOA which will optimise an objective function within a given time.

3.3. The resources

We will define 3 types of resources in a way which is convenient for this problem.

The renewable resource : most units or buildings are renewable resources. For example a building can be used to produce units, when the process ends the building is available for an other task. It is the same for units, when they end the construction of a building they are available for an other task. Unlike classical RCPSP, the number of such resources is not fixed at all, and depend on the activities already processed by the player.
The non renewable resource : In most RTS games, there is only one resource of this type, which is used to limit the total population of units. For example, it can be the total number of farms you control which determines your maximum population limit. That resource is a non renewable one since, once it is affected, it cannot come in use for something else. Again here, the number of such resources is not fixed, and depends on the activities already processed by the player.

The resource reservoirs (also called cumulative resources) : In most real time strategy games, material has to be collected on the map. It can be gold, food, wood and so on. Sometimes, the total available amount is limited, for example a gold mine may come to an end. Sometimes it is not, for example a forest may provide wood forever. We will consider that they are always available, which is the case at the start of the game. These materials are stocked in reservoirs, generally there is one reservoir for each type of material, but their capacity has no limit. These resource reservoirs are filled over time with materials at a speed which depends on the number of resources allocated by the player to that task. Materials are collected by all collector units, except when they have something else to do, such as building something. When more than one resource reservoir is present, the player can balance the number of units attributed to each one. A good assumption for the game model is to state that collectors are always collecting materials when they don’t perform a different known activity. A special building is sometimes needed to be able to collect peculiar materials, for example a farm to collect food. To increase the collecting speed, you just need to create more collecting units. The collecting speed can also be improved if you upgrade some units, or if you develop some special abilities, which of course comes at a cost. Sometimes the maximum collecting speed cannot exceed a threshold given by the game state. The key parameter for materials is the time needed to collect them. Resource reservoirs are depleted by most activities done by the player, such as produce a unit or a building.

Most activities need resources and materials in an amount which may vary, depending on the state of the game. Both renewable, non-renewable resources and materials in resource reservoirs have to be available when an activity starts. As one can see, the production model for resources can be relatively complex and is specific to each game.

3.4. The dependence-feasible graph

In such a game, it is not possible to perform any activity at any time. You have to follow a precedence constraint directed acyclic graph to construct buildings, units or upgrades. For example, as shown in table I in the ‘Starcraft’ game, you can’t build an airport if you don’t have a factory, you need barracks to produce a soldier, and so on. Often you have more than one dependence link. In some cases, you can cluster two units to make a bigger one, or you can transform some units into buildings. Again, this makes the model quite complicated.

3.5. Time constraint

Each activity, such as to construct a building or to produce a unit, takes some time to be completed. An activity cannot be stopped before the end, it can be cancelled in the game, but we won’t consider that case.

Most of the time, to produce a unit, you need a specific building which has to be available at the start of the activity. For example, to produce a marine you need barracks for 20 seconds, meanwhile the barracks cannot be used for something else. Since building can produce units only one by one, time is also a strong constraint. Buildings are also needed to perform upgrades, but they are usually effective for all units when finished.

The rule is that, to perform an activity, you have to wait for the resources to be available. An activity is thus feasible when :

- the dependence graph is respected, or will be respected when some previous activities comes to an end.
- Needed renewable resources are ready, or will be ready when some previous activities comes to an end.
- Needed non renewable resource are available, or under production.

- Needed materials are available in resource reservoir, or at least one unit is able to collect them.

When an activity is feasible, it may take some time before it actually starts, because materials may have to be collected, a building under construction may have to be finished first, or a building needed is already doing something which has to be terminated first. Waiting for resources to be available introduces lags. This decreases the efficiency of the project ; a good player will thus try to minimize delays.

3.6. The player goal

To win the game, you must eradicate your opponents, and this is the purpose of most units. Each has some hit points and some attack points. The more hit points it has, the tougher it is, the more attack points it has, the more hit points it can get from each enemy shot. Units may also have some defence points to reduce enemy attacks, some air attack points to attack air units, and so on. Some advanced units have special abilities which make them efficient, but these peculiarities may safely be ignored. We will not discuss all subtleties of these games in this paper and we will only consider that attack point is a good thing to maximize. The sum of attack points of all units is the army strength.

Considering that there are three ways to optimise an schedule , or a sequence of activities (SOA) according to the player goal :

- The army strength is given and we want the SOA which will achieve this strength in the shortest possible time. Although it is a usual RCPSP, it is not a usual situation for the player.
- The number of activities is fixed and the objective function is the army strength. This is not a good choice, because it does not take into account the duration to achieve the SOA.
- The total duration is given and the objective function is the army strength obtainable within that duration. This is a usual situation for the player.

We choose that third option because players frequently decide on a given duration without attack, it can be 5 to 20 minutes. A sequence optimised for 5 minutes will be very different from one optimised for 10 minutes, because in the latter you need to spend more on production equipments, and you don't have time for that in the first case. To take this into account, we simply evaluate the army strength obtained with all the activities in the sequence which start before the specified date. The objective function will be that army strength.

3.7. Comparison between the robot model and the game model for the scheduler

The two model are very similar. Both have renewable, non renewable and reservoir resources. The latter are filled or emptied over time depending on activities. The number of resource reservoirs are fixed in both models. One little difference is that, in the game, the number of renewable resources varies significantly, which is not the case in the robot model. The most important difference is that, in the game model, the resource reservoirs are never considered to be full, which is obviously not the case with the battery or the memory. In the game, the constraint comes from empty tank, as well as for the battery in the robot. The full memory is a constraint for the robot.

Table 1. resources comparison between models

	Robot model	RTS Game model
Renewable resources	Experiment modules	Some Units, most buildings
No renewable resources	Chemical product dose Collected samples	Some buildings (or units) for population control
Resource reservoir	Battery memory	Material tanks

4. The use of the simulated annealing algorithm for schedule optimization

The problem described above is a RCPSP which is strongly NP hard and its size is quite important. All three well known meta-heuristic methods : simulated annealing [3][4](Boctor, 1996), (Bouleimen and Lecocq, 1998), tabu search

[2](Baar & al, 1997) and genetic algorithm should be appropriate [5](Hartmann, 1998). We choose simulated annealing because it appears to be appropriate, or at least the easiest to implement. We tried the Metropolis algorithm but, as anticipated, it does not give good results. The genetic algorithm seems difficult to use because of the numerous parameters to tune. Tabu search would be nice, but it hasn't been tested for now. The simulated annealing gives good results. We have tried different version of SA, but we will describe only the best one. We will indicate some key points where a comparison with different ways was made.

Simulated annealing (SA) is a meta-heuristic which belongs to the local search algorithm class. SA is an iterative process with only one current solution and one neighborhood, in which we pull the next solution. solutions are a Sequence Of Activities (SOA) in our case. The sequence is a mere starting-time ordered activity list. The algorithm starts by evaluating an initial SOA, which is the first parameter to choose, and an SOA near the initial SOA called the neighbor. By "evaluate" we mean compute the objective function for that SOA with the game model. Then it may replace the initial SOA by this neighbor with a probability which is a function of the evaluation difference and the time already spent in the search. Then it looks again for a new neighbor and so on.

We have tuned the different parameters of the simulated annealing algorithm in the software developed for this type of game. Though this was made specifically for the 'Starcraft' game, these parameters should also be appropriate for similar games. The simulated annealing algorithm is described below :

The initial SOA

Choose the initial sequence of activities. Usually even a beginner has an idea of how to play; aberrant sequences can be avoided. But in fact, we have seen that even with a very bad start, the algorithm is able to find a good outcome in a medium size problem.

The cooling scheme

Choose the initial 'temperature' parameter T_o . Roughly a value of some tenths of a per cent of the objective function of the initial state is appropriate.

Choose the descent profile of the temperature function.

A logarithmic descent should give the absolute optimum, but it is very slow. We implemented an adaptative descent profile. The temperature will decrease by a factor µ if, for a given number M of iterations, the new sequence is always accepted. The value µ is near and below 1. For example, 0.98 works well. M should be large enough, in our case one thousand to some thousands are good values.

$$T_{k+1} = \left(\begin{array}{ll} T_k & \text{if the number of consecutive accepted transitions} < M \\ & \\ \mu.T_k & \text{if the number of consecutive accepted transitions} = M \end{array} \right.$$

The probability of acceptance is given as follows : $0.5 - 0.5 \tanh [\, 2m\, (c_k - c_{k+1} - T_{k+1})\,]$
where c_k is the cost function of iteration k, and m is the slope. The slope m can be related to T in the following way : $m=m_o/T_{k,}$ to give more significance to the slope for large values of T.

The stop criterion

There are two stop criteria. First, the process is stopped when the objective function doesn't make any significant progress for a given number N of iterations. Second, the process is stopped when the total number of iterations exceeds a given value.

The neighborhood-generation mechanism

There are many ways to choose the neighborhood definition. Some serve different purposes for the user. We describe here four of them :

a) Two activities are randomly selected in the SOA and switched.

b) One activity is randomly selected in the SOA and replaced by a randomly selected activity among all available ones.

c) randomly choose one of the following : switch two randomly selected activities inside the sequence or, replace one randomly selected activity by another randomly selected one among all available activities. The probability of switching should be much higher than the one of replacing.

d) The same as the previous one except that the replacement of an action is done by choosing a new activity inside the sequence.

Neighborhood 'a' is usable when you know exactly what actions have to be selected. The initial SOA has to be appropriate for the aim. The result will be a better permutation. Neighborhood 'b' can give some results, but it is 'c' which gives by far the best results. Option 'd' need also some knowledge from the user since the initial SOA has to contain all needed activities appropriate for the aim, only the number and the positions of each type of activity vary.

The objective function

As mentioned above, the objective function can be the plain sum of all attack points (or any desirable characteristic) of units or buildings obtained, after all activities have been performed, before a given date. The question arises of the SOA which are not feasible, and probably most of them are not feasible because of the structure of the graph of the precedence relation constraint. The first idea is to give a zero value to the function in that case, but it is not very efficient because too many SOA are rejected and it may be difficult to get out of some local maxima. It is also possible to reduce the objective function by a weight depending on the number of violations. In fact, we got very good results by just skipping infeasible activities in the SOA while evaluating the objective function. When too many activities are infeasible, the feasible activities are not able to give good results, in this way it reacts as a good weight. If you have only a few infeasible activities in a good SOA, this one can stay and then evolve favorably later. It was a great improvement of the algorithm. The slight drawback is that it increases the number of activities in the SOA at the beginning of the search.

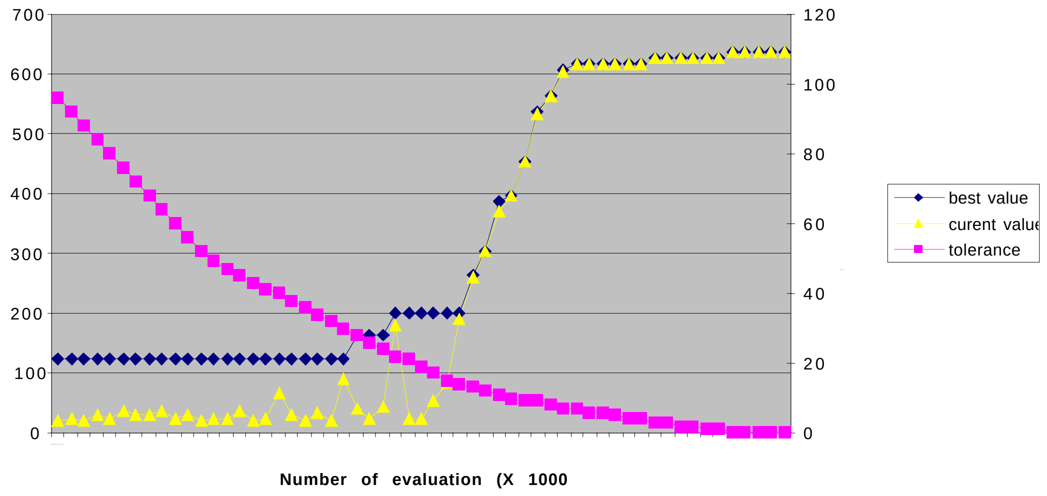

Figure 1 : Evolution of the objective function and of the tolerance (temperature) in a simulated annealing run. The initial sequence was at 128. The best sequence scores at 625.

5. Performances

The application was coded and compiled with the Builder C++. The model is so complicated that it was totally coded in C rather than using specific tools such as ILOG studio, though doing the latter is probably also possible. The tests were carried out on a PC AMD K6II-500 MHz with 128 Mbytes memory under MS Windows 98. The performances are quite good. We have compared the proposed solution with those given by an expert in a very well spread RTS game :

‘Starcraft’, and they match well. We can separate short term solutions in which we schedule about 30 activities and longer term solutions, with up to 130 activities.

For sequences with short duration, the scheduler converges to the best known solution after a few tries. For short duration, solutions are known because millions players tested them and they are available for the community on many internet sites. There are called rush play, because it is a way to win very quickly against a non prepared player, or a player who intend to prepare a long term attack. That’s why players sometimes decide in advance to avoid rush tactics.

For longer problems, small differences may arise between experienced players. The solutions are not usually published. Mainly because after some time in the game, the key factor to win won’t be only the development phase but also a good scouting of the enemy intention, to help the player in choosing a good tactic. Also some other factors such as agility interfere but this is out of our consideration. Anyway an expert player tried our solutions for longer problems and found them very satisfactory. We are not able to claim that we got the global optimum, but we can say that the best solution we found is a very good solution and even our expert is not able to find a better one. We then after compare the found solutions to our best known to determine the scheduler reliability.

Table 2 shows that you should get better than 95% of the optimum after ten tries. Surprisingly, the relative deviation decreases when the number of activities goes up. This is due to the game itself. You have more ways to achieve a good result when you have time to perform numerous activities. Consequently, the stochastic search finds its way near the optimum more easily.

Table 2. statistical results for some runs

Number of activities	27 activities	78 activities	106 activities
Project duration	450 sec	800 sec	1200 sec
Problem size	10^{38}	10^{111}	10^{151}
Optimum value for ground attack	224	824	1348
number of runs over 95% of optimum	45%	10%	10%
number of runs over 90% of optimum	45%	13.3%	30%
Average value	180	656	1207
Standard deviation	49.5	99.5	69.0
Relative deviation	18.6%	15.1%	5.7%

As we can see, the reliability of the scheduler is very good for small project. After 10 runs you have only 3 chances over 100 000 to be under 95% of the optimum. For medium-size or large problem the results are encouraging. The same work has to be done for robotic applications to determine the reliability.

On our AMD K6-II CPU, which is roughly equivalent to the best CPU used in space presently, the time to compute a small project (10 runs) is about one minute, for a longer one it is a few minutes. About the computational power available on board in the medium-term, we assume that it will be more than the present one. Then, it is obvious that the on board computational power will be largely enough for such schedulers.

Conclusion

This work is only a first try which has to be continued. The similarity between the game model and the robotic model is not due to chance. It arises because, in these RTS games, time seems to flow continuously for the player, who has to manage part of a simulated world. Then the RCPSP found here and in many human projects are similar by nature and in their complexity. Probably the game's RCPSP solved here is an easy one, because it is a game which has to be played by teenagers. But the results are still very encouraging. The transposition to a robotic scheduler, as it is described here, has to be done with different size problems. Maybe the strong constraints given by Earth visibility or day-night cycles, will be challenging. But in my opinion, these technologies cannot simply be ignored, sooner or later, they will take a prominent place in decision theory for future space robotic missions.

Acknowledgments

I Thank E. Davoust for helpful comments on the manuscript, J. Cobena for giving help in the man machine interface of the software, S. Rafin for C++ advices, M. Irondelle for expert appraisal of 'Starcraft'.

References

[1] E.H.L. Aarts and J.H.M. Korst, Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimisation and Neural Computing (Wiley, Chichester, 1989).

[2] T. Baar, P. Brucker and S. Knust, Tabu-search algorithms for the resource-constrained project scheduling problem, Technical Report, University of Osnabrück, Germany (1997).

[3]F.F. Boctor, Resource-constrained project scheduling by simulated annealing, International Journal of Production Research 34 (8) (1996) 2335-2351

[4] K. Bouleimen and H. Lecocq, A new efficient simulated annealing algorithm for the RCPSP and its multiple mode version, in: Proceedings of the 6th International Workshop on Project Management and Scheduling, Istanbul (July 7-9 1998).

[5] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Naval Research Logistics 45 (1998) 733-750.

[6]J. Jozefowska, M. Mika, R. Rozycki, G. Waligora and J. Weglarz, Simulated annealing for multi-mode resource-constrained project scheduling, Annals of Operations Research 102,135-155, (2001).

[7] R. Kolisch, A. Sprecher and A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, Management Science 41 (1995) 1693-1703.

[8] K. Neumann, C. Schwind, Project scheduling with inventory constraints, European Journal of Operational Research 56 (2002) 513-533

[9] A. Sprecher and A. Drexl, Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm, European Journal of Operational Research 107 (1998) 431-450.

SOMMAIRE/*SUMMARY*   Data Processing