Monte-Carlo Bus Regulation

Tristan Cazenave[1], Flavien Balbo[1,2], Suzanne Pinson[1]

[1] University Paris-Dauphine - LAMSADE, Place du Maréchal de Lattre de Tassigny F-75775 Paris 16 Cedex, France.
{cazenave,balbo,pinson}@lamsade.dauphine.fr

[2] INRETS - GRETIA team, avenue du Général Malleret-Joinville, F-94114 Arcueil Cedex, France.
balbo@inrets.fr

Abstract—**In this paper we want to minimize passengers waiting times at the bus stops by making buses wait at a stop. We compare a simple rule based approach to a Monte-Carlo method for this problem. When allocated enough time, the Monte-Carlo method gives better results. If the passengers arrivals and the bus travel times are known, the best algorithm is nested Monte-Carlo search with memorization which clearly outperforms nested Monte-Carlo search without memorization as well as Monte-Carlo and rule based regulation.**

I. Introduction

The development of surface public transportation networks is a major issue in terms of ecology, economy and society. To improve its attractiveness, the urban networks must increase their quality in terms of punctuality and vehicle frequency while at the same time they must decrease management costs. A project like the Bus Rapid Transit shows the benefits of improving infrastructures; but better management of the available resources is less costly than improving network infrastructures. Intelligent Transportation Systems[1] (ITS), based on synergy between new information technologies for simulation, real-time control, and communications networks are an alternative to improve available resource management. Urban traffic control systems are ITS enabling a better real-time management of available resources. The usability and the effectiveness of the urban traffic control systems greatly depends on their ability to locate, assess and react to traffic disturbances.

In order to automate the transportation activity, the theoretical bus supply is computed. It gives the transportation plan which represents the optimum supply in a theoretical context. It may become obsolete as the urban traffic conditions evolve. Regulators (the staff in charge of monitoring the bus networks) have to ensure the success of the transportation plan, in the sense of adapting theoretical supply to satisfy the passenger demand according to the urban traffic disturbances. Regulators use urban traffic control systems known as Automatic Vehicle Monitoring (AVM) systems in order to collect and display data. The use of an AVM system is the first step to the computerization of the transportation network activity. However this system is limited to detecting disturbances linked to unanticipated demands and to traffic conditions but is not able to deal with difficulties related to the real-time management of the bus network: managing the inconsistencies of data coming from sensors that locate the vehicles, assessing a disturbance according to its context as well as proposing feasible solutions. These limits are due to the inadequacy of the data collecting, shaping and displaying processes. To cope with these limits, we have proposed to complete the AVM system with a Decision Support System (DSS) in order to analyze the data so as to give a dynamic and contextual assessment of the disturbances in real-time, as well as action planning and decision making aid. The integration of an AVM system and a DSS is what we have called a Transportation Regulation Support System (TRSS). We have developed a TRSS prototype called SATIR (Système Automatique de Traitement des Incidents en Réseau - Automatic System for Network Incident Processing) [1], [2] that was tested on the Brussels transportation network (STIB). SATIR is based on a multi-agent paradigm which opens perspectives regarding the development of new functionalities to improve the management of a bus network. In this paper, we propose a new approach for bus regulation i.e. new heuristics based on Monte-Carlo simulations.

Section II describes real-time management of urban transportation networks and underlines their advantages and limits. Section III presents the different algorithms we have used. Section IV presents our experimentation and conclusions are drawn in section V.

II. Notions of the Domain

A. The Automatic Vehicle Monitoring System (AVM)

In urban transportation control domain, human regulators are located in a control center. They have to manage the transportation network under normal operating conditions (where are the buses located?) and also under disturbed conditions (where are disturbances - bus delays, bus advances - located?). What action has to be taken to solve the problem?

In most networks, vehicles are located through sensors which provide real-time information. This information represents a huge amount of data (for example data arrives every 40 seconds in the STIB network). Furthermore it may be incomplete (a sensor may break down) or uncertain (the quality of the data may be poor). This data is collected through the Automatic Vehicle Monitoring system (AVM). The AVM system compares the actual positions of the vehicles (captured by the sensors) with their theoretical positions given by pre-registered timetables in order detect disturbances representing by alarms on the screen (color code in figure 1). In this way, the regulator can see whether the vehicles are running ahead of timetable or are

[1] http://www.ewh.ieee.org/tc/its/

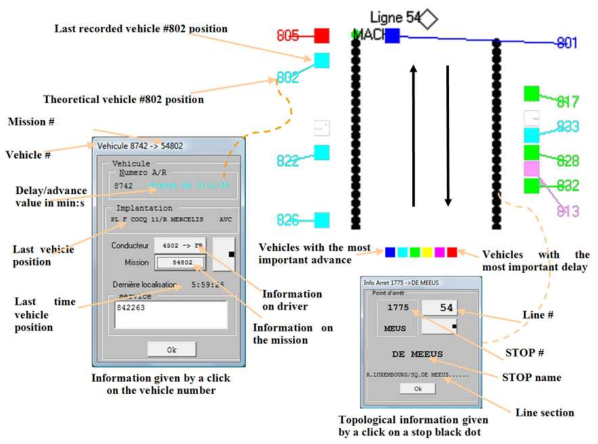
Last recorded vehicle #802 position
Theoretical vehicle #802 position
Mission #
Vehicle #
Delay/advance value in min:s
Last vehicle position
Last time vehicle position
Information given by a click on the vehicle number
Vehicule 8742 -> 54802
Vehicule
Numero A/R
8742
Implantation
PL F COCQ 11/R MERCELIS AVC
Conducteur 4302 -> FR
Mission 54802
Dernière localisation 5:59:25
service
842263
Ok
Ligne 54
MACH
805
802
822
826
801
817
833
828
832
813
Vehicles with the most important advance
Vehicles with the most important delay
Information on driver
Information on the mission
Info Arret 1775 -> DE MEEUS
Point d'arrêt
1775
54
MEUS
DE MEEUS
R.LUXEMBOURG/SQ.DE MEEUS......
Ok
Line #
STOP #
STOP name
Line section
Topological information given by a click on a stop black dot

Fig. 1. The AVM interface

running late.

Fig. 1 shows the AVM management of real-time information coming from sensors and the output of the system. Each line is represented two ways with its stops and its running buses. Each bus location is represented by 1) a number for its theoretical position coming from theoretical timetable, 2) a colored square for its real location detected by the system. This real location may be erroneous due to sensor break downs. Stops are represented by black dots. The gap between the theoretical position and the real position gives an information about the bus delays or advances. Colors give the importance of the delays or advances. Some AVM systems incorporate geographical criteria such as delay/advance alarms in a town-center, time criteria such as detecting a delay on the next scheduled departure. The role of the AVM system is to compute online basic information, organizes data collecting and displaying and computes alarms. Its screen interface facilitates the access to this data by allowing to click on a bus number to get more information on this bus or to click on a given stop to get more information on this stop (see fig. 1). Given this information, the regulators have to rely on their own experience to decide upon the regulation actions to be taken.

B. *Regulation process issues*

When a disturbance occurs, the real-time regulation process implies a planning process [2]. The aim of this planning process is to adapt the theoretical supply to the real evolution of the demand. According to the assessment of the disturbances, the regulators decide and apply different regulation procedures. The theory of the domain proposes different rules (called logic) for the regulation of a network and they underlie the computation of the timetable. If the description of these logics is not in the scope of this paper, remember that each of them is related to a specific objective. For instance, the logic of regularity aims at minimizing passengers waiting time and sharing passengers between all the buses on the line. On the contrary, the logic of taking away aims at responding to a local important demand. In this case, the timetable is computed in order to concentrate the resources on the most critical points in the line in order to keep all the passengers. Consequently, when there is a disturbance, the regulation process should not be based on the theoretical state that the timetable represents but on its objectives.

In order to support the regulator during the planning process, several regulation procedures have been studied by analytical models. A good review can be founded in [9]. However, these models aim at finding optimal solutions and are based on idealized hypotheses and simplistic networks that do not correspond to realistic problems. For example, they often assume that vehicle dwell times at stops are modeled explicitly as a linear function of the number of boarding and/or alighting passengers [11]. But in case of disturbance, the number of passengers waiting at a stop can exceed the vehicle capacity [10] making false the vehicle dwell time linear hypothesis.

To tackle these problems an in order to get a more realistic model, we have proposed a decision support system called SATIR. Its original feature is that it uses the same multi-agent model to process data and to find solutions [1], [2] along the four phases of the regulation process: 1) network monitoring is processed through dynamic timetable management; 2) distributed diagnosis is based on an original model of disturbance considering the disturbance context and its evolution, 3) feasible solutions are computed taking into account the context and profiles of vehicles. In the next we detail new heuristics to be included in the SATIR planning process.

III. Real-time Regulation Algorithms

In the following we are aiming at regulating buses using a logic of regularity based on Monte-Carlo methods. However the use of Monte-Carlo methods enables to easily change the objective of the algorithms and can be adapted to handle other logics.

In this section we present the three algorithms we have tested, the rule based approach, the Monte-Carlo and the Nested Monte-Carlo approach.

A. *Rule based regulation*

The rule based approach is inspired from the observation of human managers behavior. It consists of using a simple rule to decide when to make a bus wait at a stop. Usually, such a decision is made on a real bus line when the following bus is involved in an incident.

B. *Monte-Carlo regulation*

The Monte-Carlo algorithm adresses the problem of deciding the bus waiting times when the future is unknown.

The principle of the Monte-Carlo method applied to planning problems is to simulate planning decisions randomly. After a number of simulation steps the resulting plan is scored. A possibly large number of random simulations are performed. The mean of the simulations scores is computed for each possible planning decision, a simulation counts for a decision if the decision has been taken during the simulation. The decision that has the best mean is then chosen. This kind of Monte-Carlo method has been recently very succesful in games [12], for combinatorial problems [3] and closely related methods have also been used for transportation problems [13].

Monte-Carlo bus regulation uses the bus line simulator to simulate the line after each regulation decision. When a bus arrives at a stop, the system has to decide the time it has to wait before going to the next stop. For each possible time, the system runs a number of randomized simulations. The score associated to a possible waiting time is the average of the scores of the simulations that are run with this waiting time.

In our system the goal we have used is to minimize the waiting time of the passengers at a bus stop. We could have used another goal, such as the global waiting time including on board waiting times. The modification of the system to reach another goal would be minimal and straightforward. The score of a simulation is the sum of all the waiting times of all the passengers. Simulations are randomized so as to take into account the possible variations of the traffic, of the incidents and of the passengers flux.

The basis of the algorithm is the *randomizedSample* function that simulates the bus line, updating waiting passengers and travel times randomly, and also choosing bus waiting times at stops randomly:

```
int randomizedSample (line)
 while not end of simulation
   for all buses arriving at a stop
     decide a random waiting time for the bus
   randomly update the number of
                           waiting passengers
   randomly update the travel times
   advance one step in the simulation
 return sum of passengers waiting times
```

The upper level function that decides the waiting times at a stop simulates *nbSimulations* randomized samples for each possible waiting times and sends back the best, there is only one line being simulated, the *tmpLine* variable is a copy of the line that simulates a possible evolution of the line:

```
int chooseWaitingTime (line, bus)
for waitingTime in 1..n
  score [waitingTime] = 0
  for i = 1 to nbSimulations
    tmpLine = line
    make the bus wait waitingTime in tmpLine
    score [waitingTime] +=
          randomizedSample (tmpLine)
  if score [waitingTime] < best score
   best score = score [waitingTime]
return best waitingTime
```

C. Nested Monte-Carlo regulation

The nested Monte-Carlo regulation algorithm adresses the problem of optimizing the waiting times when the future is known. The future can be approximated taking the most probable values for each step. Another use of Nested Monte-Carlo search could be to replace the *randomizedSample* function for each random temporary line in Monte-Carlo regulation.

The idea of Nested Monte-Carlo Search is to use simulations at the lower level in order to decide the waiting times of the simulation at the current level. It has proven very successful in games such as Morpion Solitaire [5], SameGame, Sudoku [7] and Kakuro [6]. Moreover it parallelizes very well, for example at Morpion Solitaire, speedups of 56 have been obtained using 64 cores [8]. The speedups that can be obtained depend on the number of possible moves in a position, the equivalent for bus regulation is the number of possible waiting times. There are approximately 20 possible moves at Morpion Solitaire, and we have used only 4 different waiting times, so the speedups for bus regulation will probably be less than for Morpion Solitaire. However, it also mean we can increase the number of possible waiting times at no computational cost assuming a parallel implementation.

The lowest level function just simulates the bus line, randomly choosing the waiting times:

```
int sample (line)
 while not end of simulation
   for all buses arriving at a stop
     wait a random time for the bus
   advance one step in the simulation
 return sum of passengers waiting times
```

At higher levels, for each possible waiting time, a lower level simulation is performed. The waiting time that results in the lowest score is then chosen. Moreover the best sequence of waiting times is memorized in order to replay it if no better sequence is found. For each level a complete simulation is performed. The bus waiting times of a simulation of a given level are decided using simulations at the underlying level. When an underlying simulation finds a sequence of waiting times that gives better results than the current best sequence of the current level, the best sequence is updated with the waiting times found by the underlying simulation. When all simulations of the underlying level have been performed, the bus waits during the time memorized in the best sequence. The nested function is:

```
int nested (line, level)
 while not end of simulation
   for all buses arriving at a stop
     for waitingTime in 1..n
       make the bus wait during waitingTime
       if level = 1
         score = sample (line)
       else
         score = nested (line, level - 1)
       if score < best score
         best score = score
         best sequence = lower level sequence
```

```
    wait time of best sequence
  advance one step in the simulation
return sum of passengers waiting times
```

The algorithm can be made anytime with iterative calls:

```
int iterativeNested (line, level)
  while time left
    score = nested (line, level)
    bestScore = min (bestScore, score)
  return bestScore
```

IV. Experimental Results

In our experiments, we have modeled a bus line with 70 stops (including 2 terminals) and 20 buses. Every twenty time steps, a bus leaves each terminal. Zero to five passengers arrive at each stop every time step. Going from one stop to the next takes between two and six time steps. From one step to the next, the time between two stop can stay the same, increase by one time step or decrease by one time step. There is a one percent chance that an incident happens between two stops, and in this case the bus takes five more time steps to reach the next stop.

We try to minimize the waiting time of all passengers at the bus stops. The simulation runs during 100 steps.

The score of a simulation is normalized so as to be easily compared and to fit in figures (it is divided by 1,000 and shifted 200 towards zero).

If we use a fixed waiting time of one time step at every stop, the normalized score equals 171. It corresponds to the score without regulation.

A. *Rule based regulation*

In order to improve the one time step waiting time, which corresponds to no regulation at all, we have tested a rule based approach. Human regulators use rules of thumb to regulate the bus network in case of an incident, and this can be modeled with rules.

In our experiments, the regulation rules takes two values into account. The first value is δ, a threshold on the number of stops between the current bus and the following bus. The second value is w and contains the number of time steps the bus will wait at the current stop when the next bus is more than δ stops behind. If the next bus is less than δ stops behind, the current bus only stays one time step at the current stop.

Table I gives the scores resulting from the application of a rule. Each entry of the table represents the result of a simulation where a bus waits during w time steps when the following bus is more than δ stops behind. We can conclude that the best rule is to wait four time steps when the next bus is more than seven stops behind. With this rule the score of the simulation is 164 which is better than 171, the score without regulation. A conclusion is that rule based regulation is better than no regulation.

TABLE I
Scores for different rules

	w=1	w=2	w=3	w=4
$\delta = 4$	171	192	193	199
$\delta = 5$	171	191	192	195
$\delta = 6$	171	175	176	198
$\delta = 7$	171	169	166	**164**
$\delta = 8$	171	169	167	165
$\delta = 9$	171	169	167	166
$\delta = 10$	171	170	170	170

B. *Monte-Carlo regulation*

Monte-Carlo regulation performs random simulations without knowing the future. Each time a bus stops, a number of randomized samples are performed for all possible waiting times ranging from one time step to four time steps. The algorithm computes the sum of the scores resulting from the randomized samples for each possible waiting time. The system chooses the waiting time that has the minimal sum of scores.

Figure 2 gives the distributions of the scores obtained using Monte-Carlo regulation with 100 samples, with 1,000 samples and with 10,000 samples. It is clear from the figure that using more samples is beneficial since the peak of the distribution is at 165 for 100 samples and at 154 for 1,000 samples. For 10,000 samples the peak is at 147.

On a 2.83 GHz Intel core, 100 samples take 0.03 seconds.

If we compare the score of 154 with 1,000 samples it is better than the score of 171 without regulation and it is better than the best score of 164 with regulation rules. We may conclude that Monte-Carlo regulation gives better results than rule based regulation.

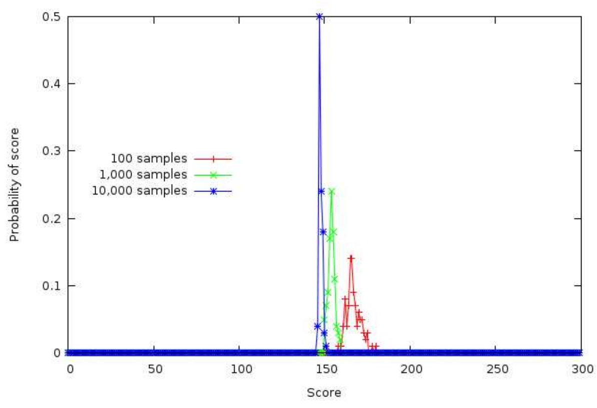

Fig. 2. Distributions of the scores for different numbers of samples.

C. *Nested Monte-Carlo regulation*

Nested Monte-Carlo regulation optimizes bus waiting times when the future distribution of passengers arrival and of travel times are known.

Figure 3 give the distributions of the scores for different levels of nesting. Each distribution is the result of 10,000 calls to the nested algorithm. A level 0 search corresponds to a sample. The peak of the distribution for a level 0 search is 210 which is much worse than all previous algorithms. We can see that a level 1 search gives better results than a level 0 search, however the peak of the distribution is at 178 which is still worse than other algorithms. A level 2 search improves a little on the level 1 search, peaking at 173 which is still a bad score.

However a level 1 search takes much more time than a level 0 search, and a level 2 search takes much more time than a level 1 search. In order to evaluate the benefits of nested calls, we computed the real time properties of the different levels using the *iterativeNested* algorithm with different levels and different time limits. The principle of the experiment is to memorize the best score found by the *iterativeNested* algorithm after each predefined computation time. The algorithm is called during the maximum allocated time, and for each predefined computation time, the best score so far is retained. The predefined computation times start at 0.01 seconds and double until 81.92 seconds. The results are given in figure 5. Each point in the figure corresponds to the mean of 1,000 calls to the *iterativeNested* algorithm. The horizontal axis is in logarithmic scale.

We can observe that for short time settings the level 0 search gives the best results, but for longer time settings it is outperformed by the level 1 search. The level 2 search is always worse than level 0 and level 1 search for search time lower than 81.92 seconds. The best mean score obtained after 81.92 seconds is 164 for the level 1 search which is better than previous results but still bad as the future is known.

In other domains where nested Monte-Carlo search was applied, it was found that memorizing the best sequence improves a lot the results of a search. We repeated the experiments for the bus regulation domain. The distributions obtained for different levels of nesting and memorization of the best sequence are given in the figure 4. Each distribution is the result of 10,000 calls to the algorithm.

The level 0 search gives the same results as without memorization, peaking at 210. The level 1 search with memorization peaks at 170 which is better than without memorization but still not a very good result compared to previous methods. The interesting results comes for the level 2 search that peaks at 65 which is much better than all previous algorithms. If we compare the distribution of level 2 with and without memorization, we can see how well memorizing the best sequence improves the nested Monte-Carlo search algorithm for bus regulation.

In order to see if a level 2 search is also good in a real-time setting, we computed the mean score of 1,000 searches of the *iterativeNested* algorithm with memorization of the best sequence at level 0, 1 and 2. The mean score found by the algorithm was computed for each predefined computation time. The results are given in the figure 6. We see in that figure that a level 1 search slightly outperforms a level 0 search and that a level 2 search gives much better results than the level 0 and level 1 searches. After 81.92 seconds of computation the mean result of a level 2 search with memorization is 52.5.

We may conclude that when the future distribution of passengers arrivals and travel time is known or when establishing a benchmark for a regulation problem, nested Monte-Carlo search with memorization is indicated.

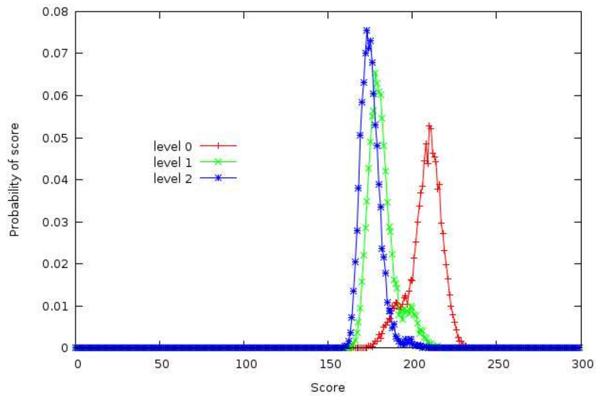

Fig. 3. Distributions of the scores for different search levels.

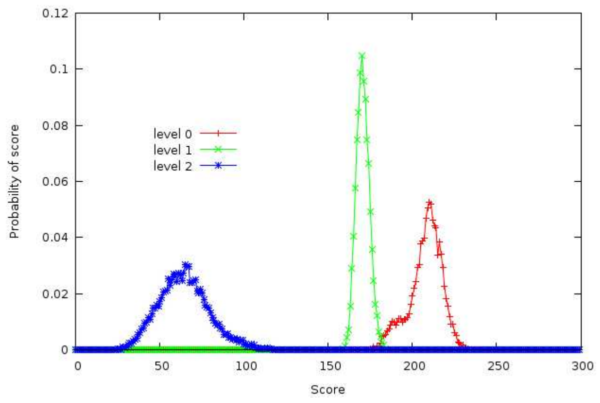

Fig. 4. Distributions of the scores for different search levels with memorization of the best sequence.

V. Conclusion and Future Work

We have presented the bus regulation problem. We have compared the simple rules approach to the Monte-Carlo method to decide bus waiting times for this problem. The Monte-Carlo method gives better results when given sufficient time. When the passengers arrivals and the bus travel times can be forecasted, the best algorithm is Nested Monte-Carlo search with memorization which clearly outperforms Nested Monte-Carlo search without memorization as well as Monte-Carlo and rule based regulation.

Future work includes improving the simulation, taking into account the number of passengers that go in and out of the bus to have a more refined minimal waiting time at each stop. We will also have to make the time taken to travel between two stops depend on the number of passengers of the bus. When there are many passengers, the traffic is supposed to be heavier.

Concerning the Monte-Carlo part, we will study the use of Nested Monte-Carlo search as a sampling strategy for randomized Monte-Carlo regulation, replacing the

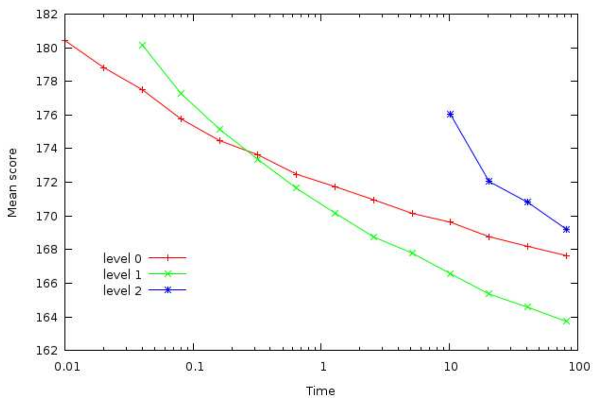

Fig. 5. Mean scores of the searches without memorization of the best sequence in a real-time setting.

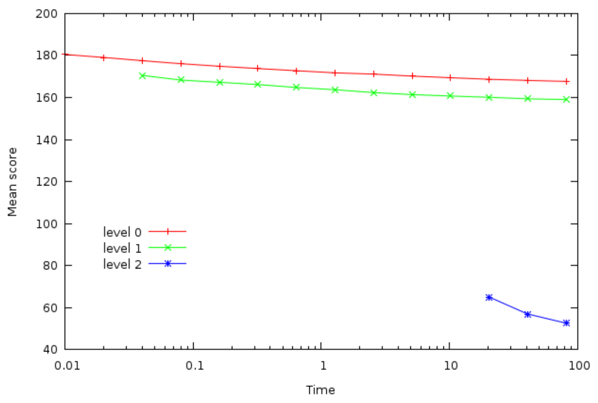

Fig. 6. Mean scores of the searches with memorization of the best sequence in a real-time setting.

call to the *randomizedSample* function with a call to the *nested* function at level 2 for example. We will also experiment with the use of nested calls for randomized Monte-Carlo regulation. Moreover possible improvements may also come from using rules in the base level simulations [4]. The computational behavior of the algorithms in a parallel architecture is also of interest since Monte-Carlo search parallelizes well.

Acknowledgments

This work has been supported by French National Research Agency (ANR) through COSINUS program (project EXPLO-RA ANR-08-COSI-004)

References

[1] Flavien Balbo and Suzanne Pinson. Dynamic modeling of a disturbance in a multi-agent system for traffic regulation. *Int. J. of Decision Support System*, 41 (1):131–146, 2005.

[2] Flavien Balbo and Suzanne Pinson. Using intelligent agents for transportation regulation support system design. *Transportation Research part C: emerging technologies*, to appear, 2009.

[3] Dimitri P. Bertsekas and David A. Castañon. Rollout algorithms for stochastic scheduling problems. *J. Heuristics*, 5(1):89–108, 1999.

[4] T. Cazenave. Playing the right atari. *ICGA Journal*, 30(1):35–42, March 2007.

[5] T. Cazenave. Reflexive monte-carlo search. In *Computer Games Workshop*, pages 165–173, Amsterdam, The Netherlands, June 2007.

[6] T. Cazenave. Monte-Carlo kakuro. In *ACG 2009*, Pamplona, Spain, May 2009.

[7] T. Cazenave. Nested Monte-Carlo search. In *IJCAI 2009*, Pasadena, USA, July 2009.

[8] T. Cazenave and N. Jouandeau. Parallel nested Monte-Carlo search. In *NIDISC Workshop*, Rome, Italy, May 2009.

[9] Guy Desaulniers and Mark D. Hickmanb. Chapter 2 public transit. *Handbooks in Operations Research and Management Science*, 14:69–127, 2007.

[10] Mahjoub Dridi and Imed Kacem. Hybrid approach for scheduling transportation network. *International Journal of Applied Mathematics and Computer Science*, 14(3):397–409, 2004.

[11] Liping Fu, Qing Liu, and Paul Calamai. Real-time optimization model for dynamic scheduling of transit operations. *Transportation Research Record*, 1857:48–55, 2003.

[12] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *ICML*, pages 273–280, 2007.

[13] R. De Leone, P. Festa, and E. Marchitto. A hybrid grasp with rollout for the bus driver scheduling. *Int. J. of Information Technology and Intelligent Computing*, 2(4), 2007.