Learning Momentum: Integration and Experimentation

J. Brian Lee, Ronald C. Arkin

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Email: blee@cc.gatech.edu, arkin@cc.gatech.edu

Abstract

We further study the effects of learning momentum as defined by Clark, Arkin, and Ram[1] on robots, both simulated and real, attempting to traverse obstacle fields in order to reach a goal. Integration of these results into a large-scale software architecture, *MissionLab*, provides the ability to exercise these algorithms in novel ways. Insight is also sought in reference to when different learning momentum strategies should be used.

1. Introduction

In 1992, Clark, Arkin, and Ram [1] presented a paper proving the validity of a concept called “learning momentum,” where the parameters determining a reactive robotic control system’s behavior are modified at runtime depending on a robot’s prior success in navigating random environments. The robot stores a short history of items such as the number of obstacles encountered, the distance to the goal, and other relevant data. The robot uses this history to determine which one of several predefined situations the robot is in and alters its behavioral gains accordingly.

Learning momentum can be considered a crude form of reinforcement learning, where if the robot is doing well, it should keep doing what it's doing and even do it more. Conversely, if it's doing poorly, it should try something different. Other examples of reinforcement learning applied to robotics include Q-learning (e.g., [3,6]), and statistical methods [5] among others.

This previous work provided a proof of concept, but more work needed to be done to fully demonstrate the utility of this approach. Until now, learning momentum had only been demonstrated in simulation; it needed to be shown viable on actual robots. Further it had only been considered as an isolated component of a robotic system. Full scale integration with a complete system architecture yet waited.

Another item of interest is when different learning momentum strategies should be used. Two strategies were previously described: ballooning and squeezing. Ballooning consists of increasing the sphere of influence (SOI) when the robot makes little or no progress towards achieving its goal. This action allows the robot to take more obstacles into account sooner (reactively) and adjust its path to navigate out of myopic situations, such as box canyons. Squeezing, on the other hand, causes the SOI to be reduced while increasing the goal's attractiveness when the robot's progress is impeded. If the robot doesn’t pay attention to nearby obstacles until they are closer, the robot has a greater chance to “squeeze” between objects that are in close proximity to each other.

This paper addresses these questions. Descriptions of experiments and their respective results are provided along with conclusions drawn from the tests.

This research is being conducted as part of a larger robot learning effort funded under DARPA's Mobile Autonomous Robotic Software (MARS) program. In this program, 5 different variations of learning, including learning momentum, are being integrated into a well-established software architecture (*MissionLab* - described in the next section). These learning mechanisms, which include reinforcement learning and case-based learning, are not only to be studied in isolation, but the interplay between these methods will be investigated as well.

2. Software Framework

The algorithms used are essentially the same as in the previous work [1] (reproduced in Appendix 1), where attention was given to three primary behaviors (schemas) [2]: *move-to-goal*, *avoid-obstacles*, and *wander*. Each behavior contributes an independent motion vector for the robot to execute. The robot weights, adds, and normalizes these vectors, and the result is transmitted to the robot for execution. The vector weights (gains) comprise three of the five parameters that are altered using the learning momentum algorithm. Another parameter is the *avoid-obstacle’s* SOI, which defines the radius of an imaginary circle around the robot. The robot reacts to objects inside the circle and ignores those outside of it. The final parameter is the *wander* vector’s persistence. Lower persistence will result in changing the *wander* direction more frequently, while a larger value will result in the same compass direction being followed for a longer period of time. One example set of adjustments made to the parameters is given in Table 1.

	Goal	Obstacle		Noise	
	Gain	Gain	Sphere	Gain	Persist.
No movement	-0.1 to 0.0	-0.1 to 0.0	-0.5 to 0.0	0.1 to 0.5	0 to 1
Progress	0.5 to 1.0	-0.1 to 0.0	-0.5 to 0.0	-0.1 to 0.0	-1 to 0
No Prog. w/ Obst.	-0.1 to 0.0	0.1 to 0.5	0.0 to 0.5	0.0 to 0.1	0 to 1
No Prog. w/out Obst.	0.0 to 0.3	-0.1 to 0.0	0.0 to 0.5	-0.2 to 0.0	-1 to 0

Table 1. Ranges of values used to alter parameters in different situations.

	Goal gain	Obstacle gain	Obstacle sphere	Wander gain	Wander Persist.
Upper	2.0	5.0	5.0	5.0	15
Lower	0.5		0.1	0.01	1

Table 2. Upper and lower bounds for parameter values.

The obstacle gain's lower limit was the sum of the current goal and wander gains. A random value was chosen within the specified range. To implement the squeezing strategy, a range of -0.5 to 0.0 was used for the SOI value in the case of "no progress with obstacles".

2.1 Adjustment Rules

After the robot has taken a certain number of steps, H_{steps}, the adjuster tries to determine what situation the robot is in. The adjuster looks at the robot's average movement, M, the average movement to the goal, M_{goal}, and the number of obstacles encountered, O_{count}. The robot's progress, P, is defined as M_{goal}/M. A movement threshold, $T_{movement}$, is defined, along with a threshold for progress, $T_{progress}$, and an obstacle threshold, $T_{obstacles}$. A robot's possible situations are defined as follows:

1. no progress with obstacles
   $M < T_{movement}$

2. movement toward goal
   $M > T_{movement}$;
   $P > T_{progress}$

3. no progress with obstacles
   $M > T_{movement}$
   $P < T_{progress}$
   $O_{count} > T_{obstacles}$

4. no progress without obstacles
   $M > T_{movement}$
   $P < T_{progress}$
   $O_{count} < T_{obstacles}$

These situations are used to choose value ranges for each parameter (see Table 1). To each parameter a random value from that parameter's respective range is then added.

2.2 Integration into *MissionLab*

All experiments described here used the *MissionLab* system developed by the Georgia Tech Mobile Robot Lab [4]. The *MissionLab* system, already possessing capabilities to task and control simulated or real robots with reactive behaviors, was expanded to include learning momentum. Test runs on simulated robots were carried out using *MissionLab*'s simulator, and the real robotic experiments described below also were controlled directly using *MissionLab*.

The integration of the new algorithms into *MissionLab* was relatively simple. First, the relevant behaviors were modified to use the robot's global variable space for parameter values instead of using hard-coded values at run-time. Functionality was then added to save relevant history information, which was mainly the robot position, distance from the goal, and the number of obstacles encountered. Once that foundation was set up, all that was required was to write a routine to extract the situation from the history, extract the parameters from the robot's global variable space, alter the values as dictated by the learning momentum rules, and replace them in the global variable space. A call to this routine was inserted at the end of the robot's cycle, so the parameters are adjusted every time the robot moves H_{steps} steps.

3. Experiments in Simulation

This section discusses the methods used in simulation testing of these ideas and their results.

3.1 Simulation Environment

In the first phase of this work, *MissionLab* was used to gather data on simulated robots to test both the integration and provide reasonable parameters for use on physical robots. *MissionLab*'s automatic obstacle generation capability was used to produce obstacle fields measuring 150m x 150m. The robot, starting at the coordinates (10m, 10m), where (0m, 0m) is the lower left corner of the field, was instructed to move 153m to coordinates (140m, 90m). Initially, four obstacle fields were created, two with 15% obstacle coverage and two with 20% obstacle coverage. All obstacles were circular and 1m in diameter. Robots using learning momentum, both squeezing and ballooning, were sent through the obstacle field at least fifty times each while their progress and position were logged. Robots without learning momentum were also sent through the obstacle field. For the non-learning robots, the goal and obstacle gains were both 1.0, the SOI was 1.0m, and the wander persistence was 10. Three series of runs were made for each obstacle field with wander gains of 0.3, 0.5, and 1.0, respectively. This was the first set of tests. See Figures 10 and 11.

For further testing, four more obstacle fields were created. The field size, start position, and end position were the same as in the earlier fields. Again, two fields were created with 15% obstacle density, and two were created with 20% obstacle density, but this time the obstacles ranged in size from 0.38m to 1.43m. Robots with the same learning momentum values were then sent through the obstacle field again to assess the difference that varying obstacle sizes would make.

Different learning momentum values were tested throughout the experiment to try and assess how their modification would affect the robot behaviors. In one instance, the wander persistence ceiling was lowered from 15 to 10, and in another, the growth of the wander gain in the "no progress with obstacles" situation was accelerated. The range of values used to change the wander gain in the "no progress with obstacles" situation was increased from [0.0, 0.1] to [0.0, 0.5]. The best resulting parameters were used in the real robotic experiments as derived from a simulated robot moving through a world patterned after the real world the physical robot would traverse.

3.2 Simulation Results

After running the simulations, two immediate results presented themselves: 1) learning momentum has the capacity to greatly increase a robot's ability to successfully traverse an obstacle field, and 2) successful completion comes at the expense of time.

Completion rates of obstacle fields with uniform obstacle sizes and varying obstacle sizes are given in figures 1 and 2, respectively. Sets A – D refer to tests where obstacle size was not changed, and sets E – H refer to tests where obstacle size varied within the environment. Also, average steps to completion are given in Figures 3 and 4.

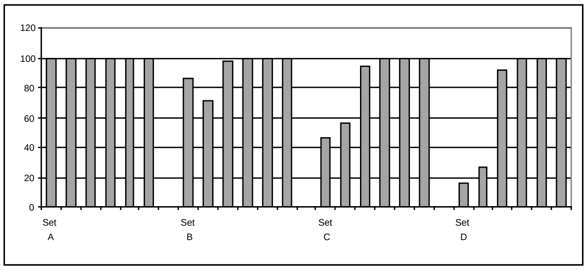

Figure 1. % complete with uniform obstacle size.

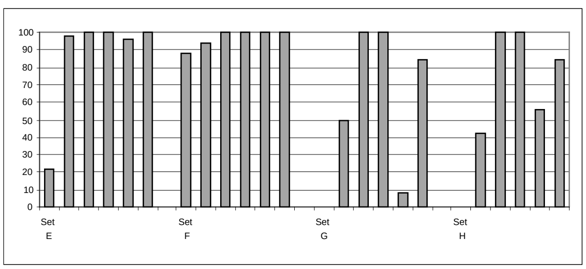

Figure 2. % complete with varying obstacle sizes.

	LM Strategy	Wander Gain	Wander Upper Limit
Bar 1	None	0.3	NA
Bar 2	None	0.5	NA
Bar 3	None	1.0	NA
Bar 4	Ballooning	NA	15
Bar 5	Ballooning	NA	10
Bar 6	Squeezing	NA	15

Table 3. Differences between parameters with uniform obstacle size.

	LM Strategy	Wander Gain	Wander Delta Range
Bar 1	None	0.5	NA
Bar 2	None	1.0	NA
Bar 3	Ballooning	NA	0.0 – 0.1
Bar 4	Ballooning	NA	0.0 – 0.5
Bar 5	Squeezing	NA	0.0 – 0.1
Bar 6	Squeezing	NA	0.0 – 0.5

Table 4. Difference between parameters with varying obstacle sizes.

For sets E – H, the wander delta range corresponds to the range of values from which a random value is picked to change the wander gain in the case of "no progress with obstacles."

The first observation is that the robot has a more difficult time traversing the fields with smaller obstacles. Second, robots with learning momentum have a greater completion rate in nearly all tested situations than robots without it, but with an accompanying increase in time. The robots without learning momentum that were successful on average took much less time than robots with learning momentum.

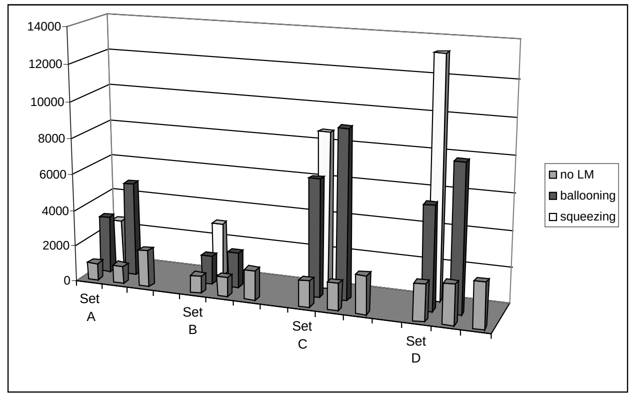

Figure 3. Average steps to completion with uniform obstacle size. Bars correspond to Table 3 and are numbered left to right, and front to back.

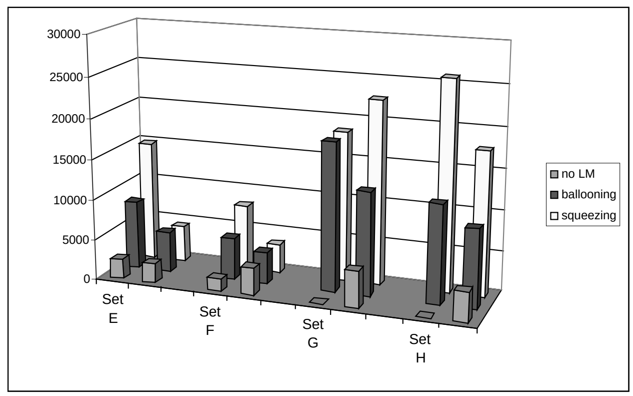

Figure 4. Average steps to completion with varying obstacle sizes. Bars correspond to Table 4 and are numbered left to right, and front to back. Values set to zero represent no successful runs.

On average the ballooning strategy took less time than the squeezing strategy. This is due to the fact that ballooning and squeezing are geared toward different situations, both of which occurred in all of the simulated environments. Ballooning works best with box canyon situations, while squeezing works better while moving between many closely spaced obstacles. While the robots did well in the situations for which their strategies were developed, difficulty arose when robots using one strategy encountered situations best suited for the other. If a robot allowed its SOI to get large enough to balloon out of a box canyon situation, then it found great difficulty moving through closely spaced obstacles. It had a tendency to settle in areas of locally less dense obstacles. If a robot using the squeezing strategy found itself in a box canyon situation, it has a hard time getting out because of its reduced SOI. On average, it takes longer for a squeezing robot to move out of a box canyon than it does for a ballooning robot to move through closely spaced obstacles, hence the average lower time for the ballooning robots.

After the simulated results had been assessed, a simulation environment was created to approximate the real robot experimental environment. As before, robots utilizing ballooning, squeezing, and no learning momentum were allowed to traverse the environment. The results are given in figures 5 and 6.

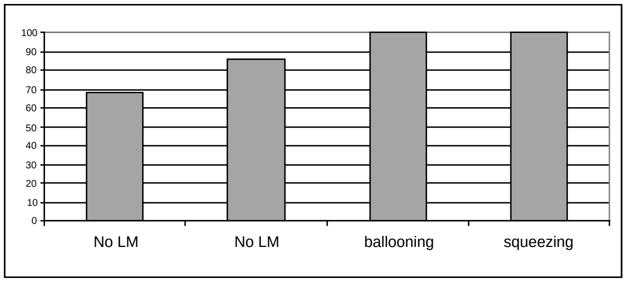

Figure 5. % complete for simulated environment

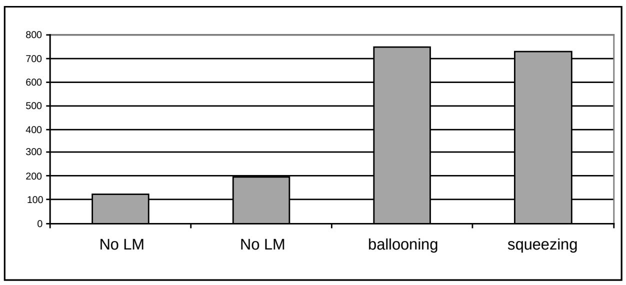

Figure 6. Average steps to completion for simulated environment

As before, an increase in completion with learning momentum occurs, but the accompanying increase in time was also present. The difference between ballooning and squeezing is very small here; this similarity probably arises from the fact that the world had a relatively low obstacle density, as shown in Figure 7.

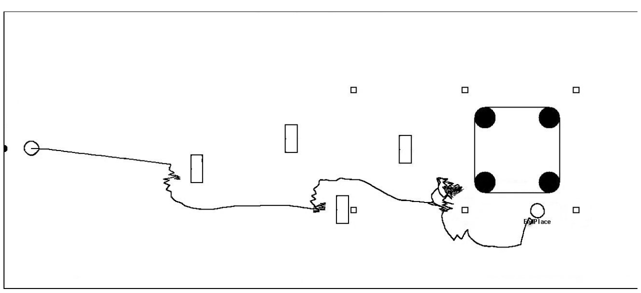

Figure 7. A sample run of a simulated real-world environment.

4. Robot Experiments

4.1 Experiment Setup

After the simulation tests, a real robot w as tested. A Nomad 150 robot was used with sonar rings for obstacle detection. The experimental area was approximately 24m x 10m with various obstacles arranged to reflect the simulated tests (see Figure 8). The robot's start place was at (1m, 5m), where (0m, 0m) was the lower left corner of the test area (see Figure 7). The MissionLab system was used to control the robots directly from off-board computers using wireless serial links.

Figure 8. A typical test run where a robot traverses a simple obstacle field similar to the one in Figure 7.

The robot was sent through the obstacles repeatedly until results from four successful ballooning runs and four successful squeezing runs were obtained. Failures when using learning momentum were determined to be caused by sonar failures instead of algorithmic failures due to the fact that the robot's safety margin was never altered and from visual representations of the sonar readings when collisions occurred. If the robot ran for ten minutes without a collision or without reaching the goal, that run was considered a failure. Four runs without learning momentum were also performed. Both runs had a value of 1.0 for the *move-to-goal* and *avoid-obstacle* gains. Two runs had a *wander* gain of 0.3 and two had a *wander* gain of 0.5. All had 3.0m SOI and 0.5m safety margin.

4.2 Experiment Results

The robots that traversed the obstacle field performed as predicted. The robots using learning momentum that didn't have sonar failures all reached the goal. None of the robots without learning momentum completed the task. This lack of any completions probably arose from the small number of trials. The results of the real tests are summarized in Figure 9.

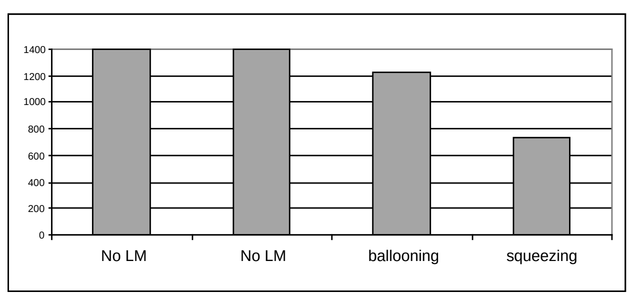

Figure 9. Average steps to completion for a real environment. Trials with no successful runs were given the largest value on the graph.

The results for squeezing were very close to the simulated results, while the ballooning took a bit longer than expected. The possibility does exist that with more test runs, the average could drop so that the difference between simulated and real results is not as large. During the experiments, the robots without learning momentum did seem to make faster progress up to the point where they got stuck, which would be consistent with the simulation results.

5. Conclusion

Learning momentum has been shown to work both in simulation and on actual robots, and it seems to have both pros and cons when applied to obstacle avoidance and navigation. It can greatly increase a robot's ability to successfully an obstacle field, but those successful runs come at a price of time. The different strategies also seem to work both for and against the robot, depending on the situation. Ballooning robots can move around obstacles and get out of box canyon situations easily, but they may get stuck in front of holes they should obviously be able to move through. Squeezing robots move between closely spaced obstacles, but they get stuck in box canyon situations. If knowledge of the terrain is available beforehand, an appropriate strategy can be chosen, but if the terrain is totally unknown, ballooning would probably be the better choice since robots using it seem to be able to overcome its problem situations a little quicker than robots using the squeezing strategy.

Future work focuses on allowing the robot to choose appropriate strategies at run time and to recognize when large changes need to be made to its parameters. Case-based reasoning or a higher level set of learning momentum rules could be used, and if successful, the time it takes for a robot to negotiate an obstacle field should be reduced significantly without sacrificing success rates.

Acknowledgments

This research is supported under DARPA's Mobile Autonomous Robotic Software Program under contract #DASG60-99-C-0081. The authors would also like to thank Dr. Douglas MacKenzie, Yoichiro Endo, Alex Stoytchev, William Halliburton, and Dr. Tom Collins for their role in the development of the *MissionLab* software system.

References

[1] Arkin, R.C., Clark, R.J., and Ram, A., "Learning Momentum: On-line Performance Enhancement for Reactive Systems", Proceedings of the 1992 IEEE International Conference on Robotics and Automation, May 1992, pp. 111-116.

[2] Arkin, R.C., ``Motor Schema-Based Mobile Robot Navigation", International Journal of Robotics Research, Vol. 8, No. 4, August 1989, pp. 92-112.

[3] Asada, M., Noda, S., Tawaratasumida, S., and Hosoda, K., "Vision-Based Reinforcement Learning for Purposive Behavior Acquisition", Proc. IEEE International Conference on Robotics and Automation, May 1995, pp. 146-53

[4] MacKenzie, D., Arkin, R.C., and Cameron, R., ``Multiagent Mission Specification and Execution", Autonomous Robots, Vol. 4, No. 1, Jan 1997, pp. 29-52.

[5] Maes, P. an d Brooks, R., "Learning to Coordinate Behaviors", Proc. AAAI 1990, Boston, MA, August 1990, pp. 796-802.

[6] Mahadevan, S., and Connell, J., "Automatic Programming of Behavior-based Robots Using Reinforcement Learning", Proc. 9th AAAI, Anaheim, CA, July 1990, pp. 768-773.

Appendix 1 - Pseudocode for Learning Momentum Algorithm

```
M = movement threshold
P = progress threshold
Obs = obstacle threshold

steps is initialized to zero the first time

if ( steps == HISTORY_INTERVAL )
    calculate the average movement and progress of the
      robot

    motion ratio = progress / total movement
    if ( average movement < M )
        situation = no movement
    else if ( motion ratio > P )
        situation = progress
    else if ( obstacle count > Obs )
        situation = no progress w/ obstacles
    else
        situation = no progress w/out obstacles

    for each learning momentum parameter
        get the range of numbers used to alter the
          parameter given the current situation
        get a random number within the range
        add the random number to the parameter

    steps = 0

steps++
```

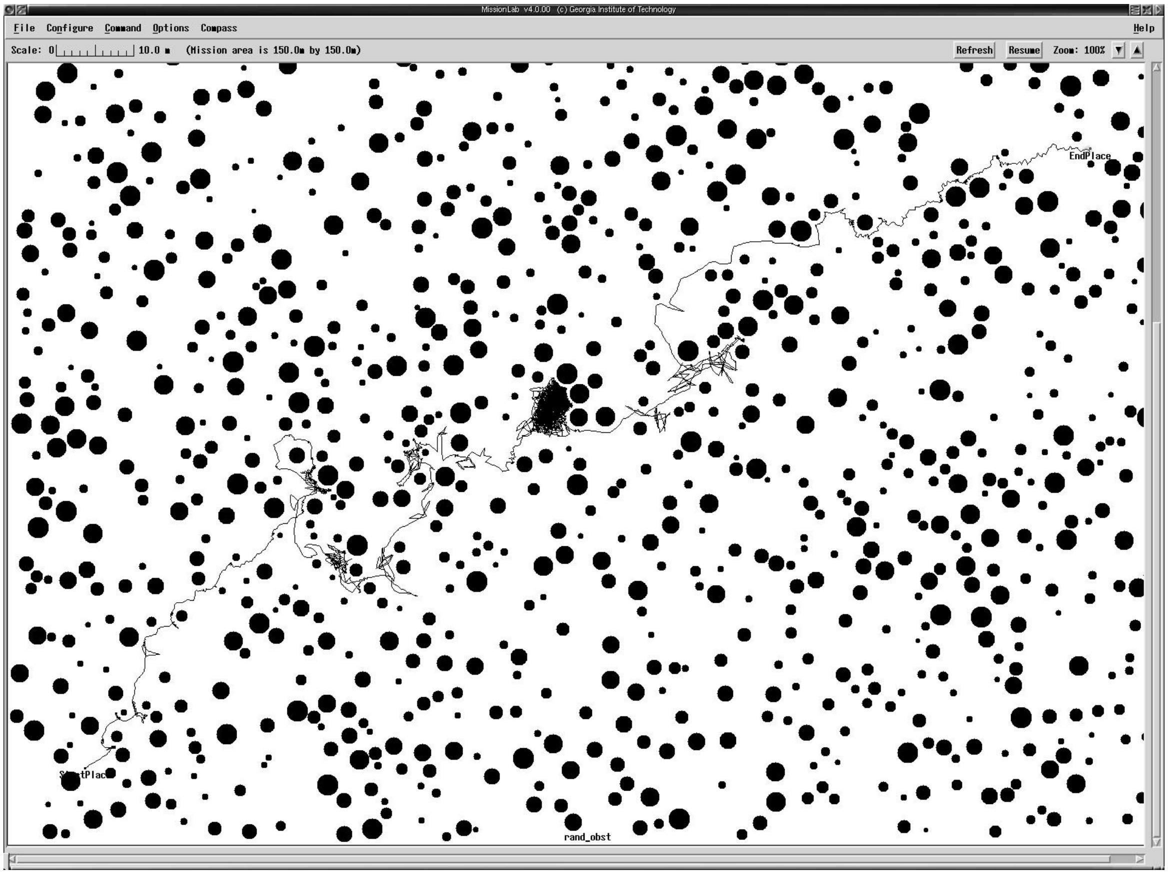

Figure 10. 15% coverage - ballooning

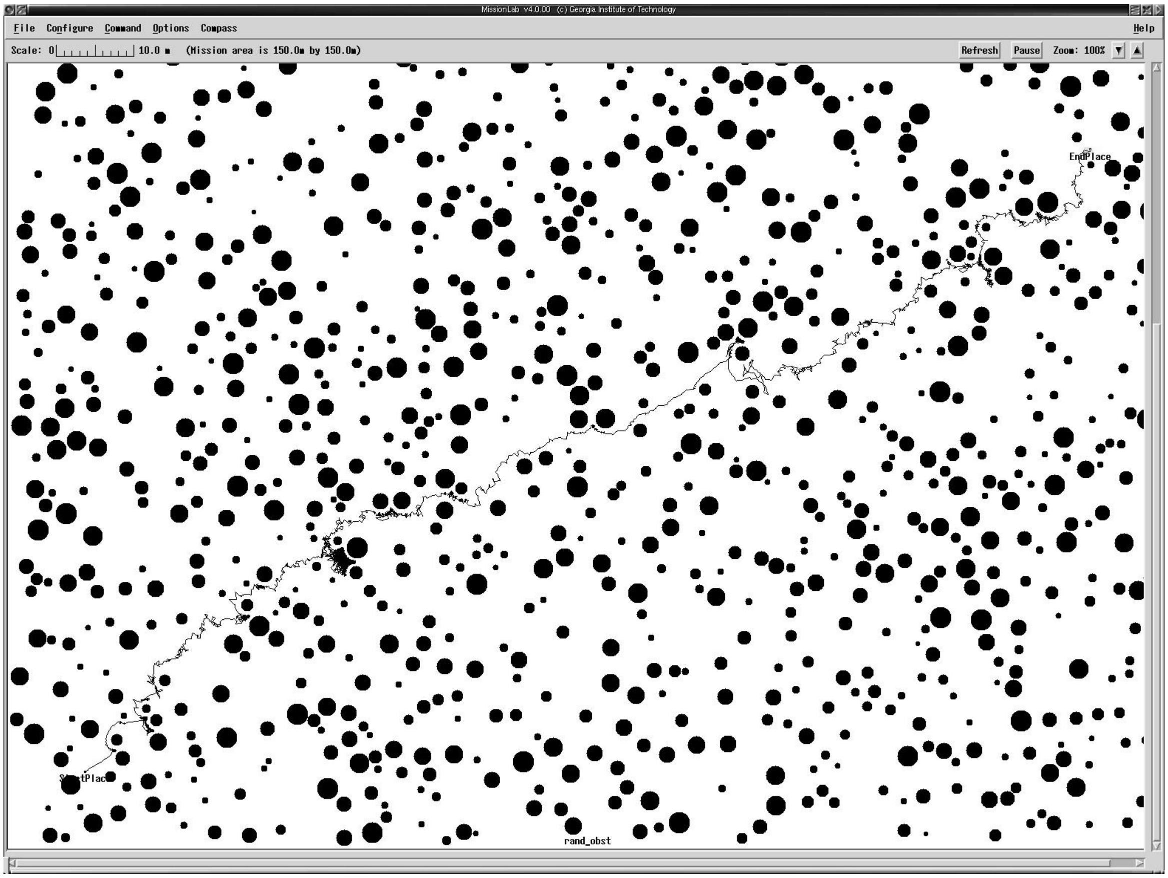

Figure 11. 15% coverage - squeezing