A Decision Support System for Crowd Control

Johan Schubert, Luigi Ferrara, Pontus Hörling, Johan Walter
Department of Decision Support Systems
Division of Command and Control Systems
Swedish Defence Research Agency
SE-164 90 Stockholm, Sweden
schubert@foi.se
http://www.foi.se/fusion

Abstract

In this paper we describe the development of a decision support system for crowd control. Decision support is provided by suggesting a control strategy needed to control a specific current riot situation. Such control strategies consist of deployment of several police barriers with specific barrier positions and barrier strengths needed to control the riot. The control strategies are derived for a set of pre-stored example situations by using genetic algorithms where successive trial strategies are evaluated using stochastic agent-based simulation. The optimal control strategy for the current situation is constructed by first finding the best linear combination of pre-stored example situations. The optimal strategy is given as the same linear combination of associated strategies.

Keywords: Crowd control, riot control, decision support, learning, simulation, fuzzy measure, genetic algorithms.

Introduction

In international peace-keeping operations today, our forces must be able to handle riots and successfully control crowds. This makes it important to derive control strategies for different riot situations (Grieger 2000). In this paper we describe the development of a decision support system for crowd control. Decision support is provided by suggesting a control strategy needed to control a specific current riot situation. Such control strategies consist of deployment of several police barriers with specific barrier positions and barrier strengths needed to control the riot. These crowd control strategies may be evaluated using a riot simulator and used to provide decision support when found effective (Schubert & Suzić 2007). The tactical commanders, responsible for keeping security, may use those control strategies derived for simulated situations that are most similar to the on-going situation, and study the predictive situation picture given alternative courses of actions.

The optimal control strategy for the current situation is found by comparing the current situation with pre-stored example situations of different sizes. The control strategies are derived for these pre-stored example situations using Genetic Algorithms (Holland 1973) where strategies are evaluated using stochastic agent-based simulation. Each strategy corresponds to the collected information about the strength at every possible predetermined barrier position. The simulations are used to generate a data base over simulated riot cases in a certain city environment. This is

in Proc. 13th Int. Command and Control Research Tech. Symp.,
Track 3, Paper 005, pp. 1-19, Seattle, WA, USA, June 2008,
US Dept. of Defense CCRP, Washington, DC, 2008.

done beforehand for a specific city or suburb using its street network, before an assumed critical situation might arise.

The problems we have to solve to realize the sought-after decision support are:

- a decision theoretic matching algorithm comparing a current situation with pre-simulated situations,
- an appropriate computer representation of control strategies,
- an effective agent simulation model.

A novelty in our approach is that we combine agent based simulation and genetic learning to generate optimal ways to control a crowd.

In the last several years there has been a substantial amount of research on how to model crowds and give them complex behaviors using intelligent agents. Several scientists involved in Project Albert (2008) (US Marine Corps) have also used genetic algorithms for optimization of agent-based behavior. In a project report Graves *et al*. (2000) demonstrated how to use genetic algorithms to improve the rule bases that define when individual agents take various actions in agent-based simulation, e.g., when advancing towards enemy lines or shooting at enemy soldiers. Dixon and Reynolds (2003) used genetic algorithms for peacekeeping scenarios on their Behavior Action Simulation Platform (BASP) for cases where it is anticipated that the model will evolve from one or more preceding models. In spite of all this research there is very little work on the subject of crowd control.

In the next section we describe how decision support is designed and realized using crowd control strategies. This is followed by a section on learning the strategies for crowd control. The fourth section presents how stochastic agent-based simulation is used in the learning process. Finally, conclusions are drawn.

Decision Support

In this section we describe how decision support is designed using the results from a learning and simulation system.

Decision support is provided to the tactical commander as the optimal control strategy for the current situation. It is found by comparing the current situation with pre-stored example situations of different sizes, Figure 1. With each of these pre-stored situations an optimal control strategy is associated. The current riot situation may have a best match to a superposition of a subset of these pre-stored situations. The decision support will then be given as the corresponding superposition of control strategies.

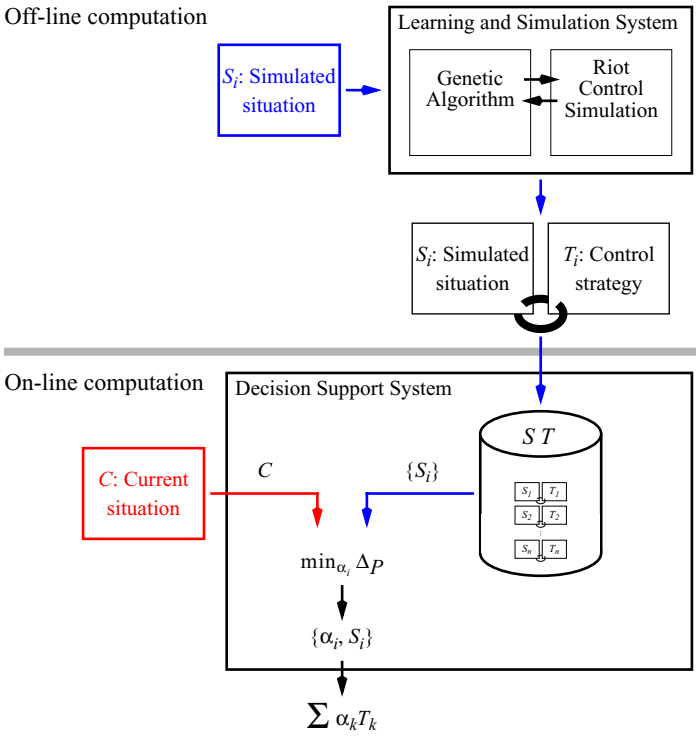

Figure 1. On-line decision support is given as $\{(\alpha_k, T_k)\}$ where $\{(\alpha_k, S_k)\}$ minimizes the distance Δ_P to the current situation C.

On-line decision support is given as $\{(\alpha_k, T_k)\}$ where $\{(\alpha_k, S_k)\}$ minimizes the distance Δ_P to the current situation C.

Let us investigate how the optimal control strategy may be constructed. Let $S = \{S_i\}$ be the set of all simulations S_i. We have $S_i = \{\mu_j^i\}$, where μ_j^i is the starting position[1] of the j^{th} agent of S_i.

For a particular simulation S_i we calculate a 2-dimensional fuzzified position map P^{S_i} around each agent starting position μ_j using a normal distribution

$$P^{S_i}_{\mu_j^i}(x_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\|x_k - \mu_j^i\|_2^2}{2\sigma^2}}, \qquad (1)$$

where $P^{S_i}_{\mu_j^i}(x_k)$ is the degree to which the j^{th} agent's starting position is x_k, $\|x_k - \mu_j^i\|_2^2$ is the Euclidean distance between the two positions and σ^2 is the variance of the distribution. The variance used is domain dependent and must be adjusted in relation to the used resolution of the position map, Figure 2.

1.It should be noted that we treat each position of a simulation run as alternative starting positions for different simulations labeled S_i.

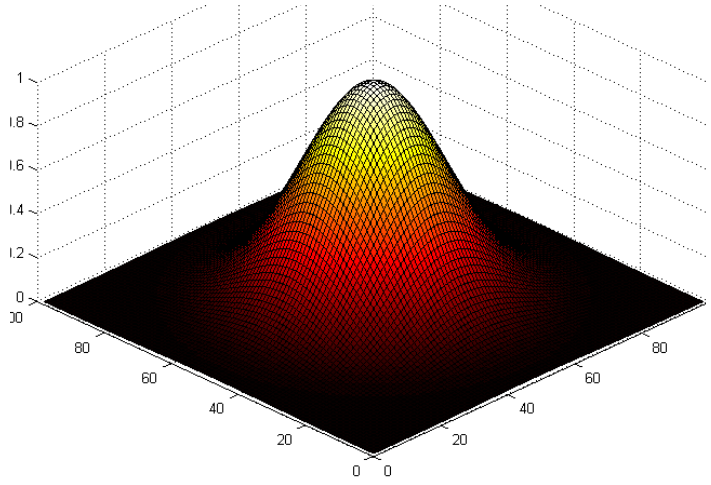

Figure 2. An example of $P^{S_i}_{\mu^i_j}(x_k)$.

For each position x_k in the map we sum up the contribution from each agent's starting position μ^i_j

$$P^{S_i}(x_k) = \sum_{\mu^i_j \in S_i} P^{S_i}_{\mu^i_j}(x_k), \tag{2}$$

see Figure 3.

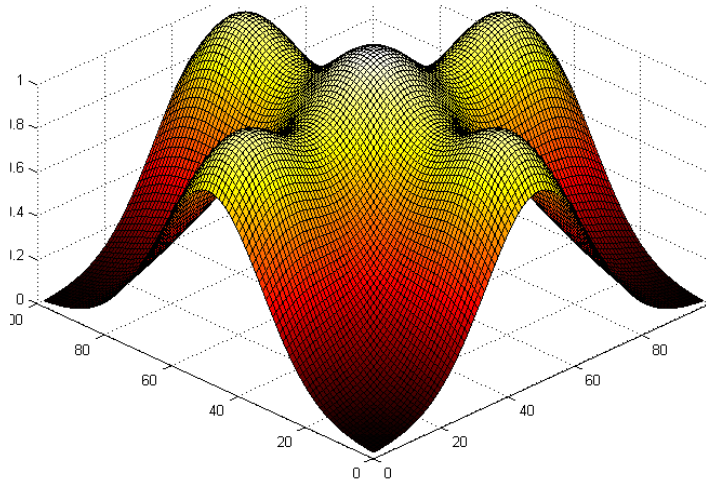

Figure 3. An example of $P^{S_i}(x_k)$.

To each simulation S_i we have a strategy T_i attached. In order to find the best decision support we compare the current situation C with all simulations S_i.

First, we calculate a 2-dimensional fuzzified position map P^C around each agent starting position μ^i_j in the current situation C, using a normal distribution in the same way as was done in Eq. (1),

$$P^C_{\mu^i_j}(x_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\|x_k - \mu^i_j\|_2^2}{2\sigma^2}}. \tag{3}$$

Here, $P^C_{\mu^i_j}(x_k)$ is the degree to which the j^{th} agent's starting position in the current situation C is x_k.

As in Eq. (2) we sum up the contributions from each agent's starting position μ_j^i for each position x_k in the map

$$P^C(x_k) = \sum_{\mu_j^i \in C} P^C_{\mu_j^i}(x_k), \tag{4}$$

see Figure 4.

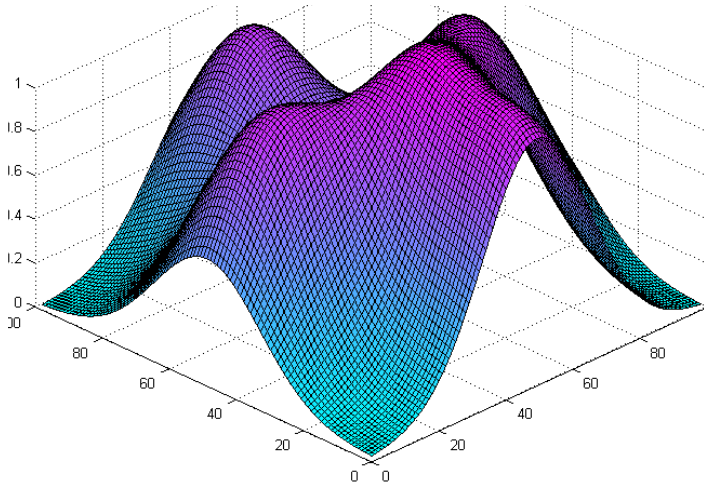

Figure 4. $P^C(x_k)$.

Using P^C we can now find the best decision support. We calculate the sum of absolute differences between P^C and the optimal linear combination of all P^{S_i} for all positions x_k in the map

$$\Delta_P(P^C, \{P^{S_i}\}) = \sum_{x_k} \left| P^C(x_k) - \sum_i \alpha^i P^{S_i}(x_k) \right| \tag{5}$$

see Figure 5.

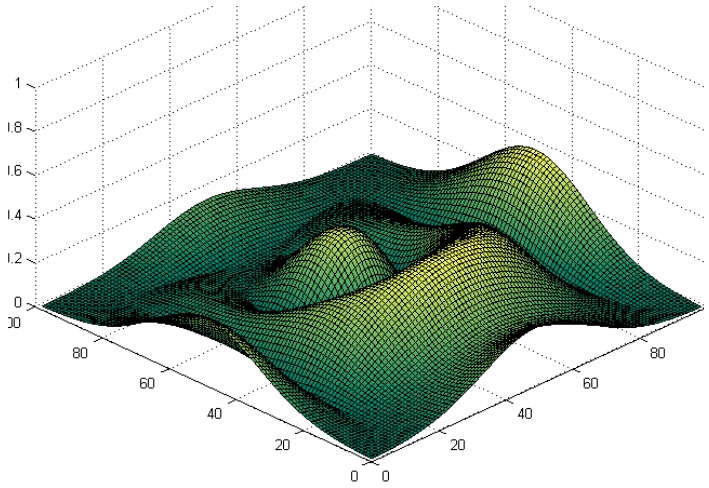

Figure 5. $P^C(x_k) - \sum_j \alpha^j P^{S_j}(x_k)$.

In order to facilitate optimization we will exchange Eq. (5) with

$$\Delta_P(P^C, \{P^{S_i}\}) = \sum_{x_k} \left[P^C(x_k) - \sum_i \alpha^i P^{S_i}(x_k) \right]^2 \tag{6}$$

where the absolute value of Eq. (5) is exchanged for a quadratic that corresponds to the same optimum. This gives us a quadratic optimization problem.

Decision support is given as the linear combination of all strategies $\sum_i \alpha^i T_i$ whose corresponding linear combination of simulations S_k minimizes the difference between the fuzzified position-map P^C and $\sum_j \alpha^j P^{S_j}$.

Whenever several control strategies in $\{T_k\}$ have strength greater than zero for some barrier position their weighted strengths are simply summed up as the decision support regarding that particular barrier.

Matching Algorithm

A *simulation* is defined as a set of starting positions and a barrier configuration that best meet the attack of agents starting at those positions.

An *observation* is a set of starting positions for the observed agents that is about to attack our key building. This is our current situation.

We need to find those simulations that best match the observation and merge those barrier configurations.

We start by filtering out those simulations that overlap the observation. A simulation overlaps an observation if any position in the simulation overlaps any observed position. A simulated position overlaps an observed position if its truncated normal distribution overlaps. We let σ denote the standard deviation and cut off the distribution at 95% which leads to a radius of:

$$r = 1.95996 \cdot \sigma \tag{7}$$

We choose the value of σ to be half the average distance between two observed positions.

If the distance between a simulated position and an observed position is less that 2 * *r*, their distributions overlap, Figure 6.

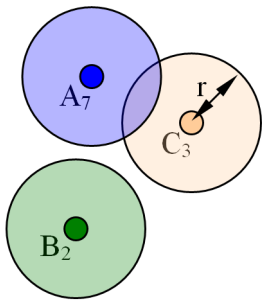

Figure 6. If the distance between a simulated position and an observed position is less that 2 * *r*, their distributions overlap.

The figure shows how the observed position C_3 overlaps with position A_7 from simulation A and does not overlap with position B_2 from simulation B. So the observation overlaps with simulation A and not B.

To quickly access all simulated positions close to an observed position, all simulated positions are grouped in a grid. The grid size is $2r * 2r$, Figure 7.

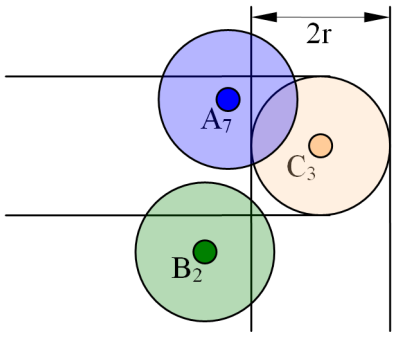

Figure 7. To quickly access all simulated positions close to an observed position, all simulated positions are grouped in a grid. The grid size is $2r * 2r$.

To find all simulated positions that overlap with an observed position, we have to check the simulated positions in the same grid square as the observed position, and the eight surrounding grid squares.

Consider the simplified problem of matching an observation, C, of five positions, depicted as brown dots in Figure 8, against two simulations, A and B, depicted as blue and green dots, both containing five observations.

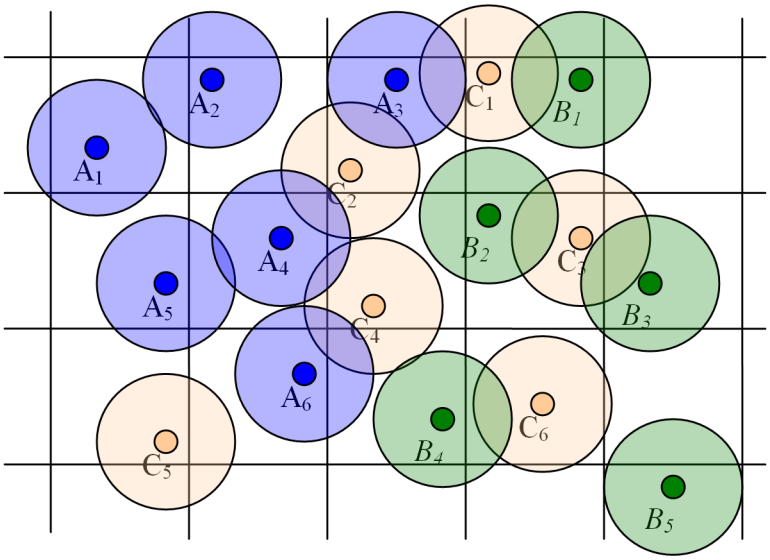

Figure 8. Observed positions, C, and how they overlap with the simulated positions A and B.

To find which simulations that overlap with the observation, we proceed as follows. Start with C_1. We have B_1 in the same grid square and A_3, B_2 and B_3 in the surrounding grid squares. Of these only A_3 and B_1 overlap with C_1. We do the same for C_2 to C_6. We find that C_1, C_2, C_3, C_4, C_6 overlap positions A_3, A_4, A_6 and B_1, B_2, B_3, B_4. Hence simulations A and B overlap observation C, since at least one of their positions overlap.

We create a normal distribution for the overlapped observation positions and the overlapped simulation positions: P^C, P^a and P^b, all being m x n matrices. We need to decide how much of A and B we need that best matches C. That is, we need to minimize the function according to α, which is an n x 1 matrix

$$f(\alpha) = \sum_{i,\,j} [P^C_{ij} - (\alpha^a P^a_{ij} + \alpha^b P^b_{ij})]^2 \tag{8}$$

Or generally

$$f(\alpha) = \sum_{i,\,j} \left[P^C_{ij} - \sum_{s \in S} \alpha^a P^a_{ij} \right]^2, \tag{9}$$

where S is the set of all simulations.

This function can be rewritten on the format:

$$f(\alpha) = R\alpha + \alpha^T Q\alpha\,, \tag{10}$$

where R is a 1 x n matrix and Q an n x n matrix.

Using,

$$\begin{aligned} R_s &= -\sum_{ij} (P^C \times P^s)_{ij} \\ Q_{s,\,s} &= \sum_{ij} (P^C \times P^s)_{ij} \\ Q_{s,\,t} &= 2\sum_{ij} (P^i \times P^j)_{ij}, \end{aligned} \tag{11}$$

where $\times$ denotes element-wise multiplication.

If Q is a positive semi-definite matrix, then $f(\alpha)$ is a convex function. In this case the quadratic program has a global minimizer if there exists at least one vector x satisfying the constraints and $f(\alpha)$ is bounded below on the feasible region. If the matrix Q is positive definite then this global minimizer is unique. If Q is zero, then the problem becomes a linear program.

This equation can be solved using the simplex method for quadratic programming (Wolfe 1959).

The reason for matching the simulations with the observation was to merge the simulations barrier instances to get the barrier instances that best meet the attack of the observed agents. We use the alphas to scale the barriers from the simulations. The strength of barrier b_i is the sum of the strength of barrier b^a_i in simulation A times α^a and the strength of barrier b^a_i in simulation B times α^b divided by the sum of the alphas,

$$b_i = \frac{\alpha^a b_i^a + \alpha^b b_i^b}{\alpha^a + \alpha^b}. \tag{12}$$

Or generally,

$$b_i = \frac{\sum_{s \in S} \alpha^s b_i^s}{\sum_{s \in S} \alpha^s}. \tag{13}$$

The Decision Support System Interface

The decision support system consists of a map, various lists in the right field and some action buttons at the bottom. The image shows the Swedish town of Norrköping, Figure 9. The yellow buildings are key buildings. The stars are assembly points. The blue lines are the police barriers the system has manned — after pressing the **Barrier Decision Support** button — in order to meet the threat of the rioters and demonstrators depicted as red dots. The number depicted on the barrier is the number of policemen in that barrier, Figure 9.

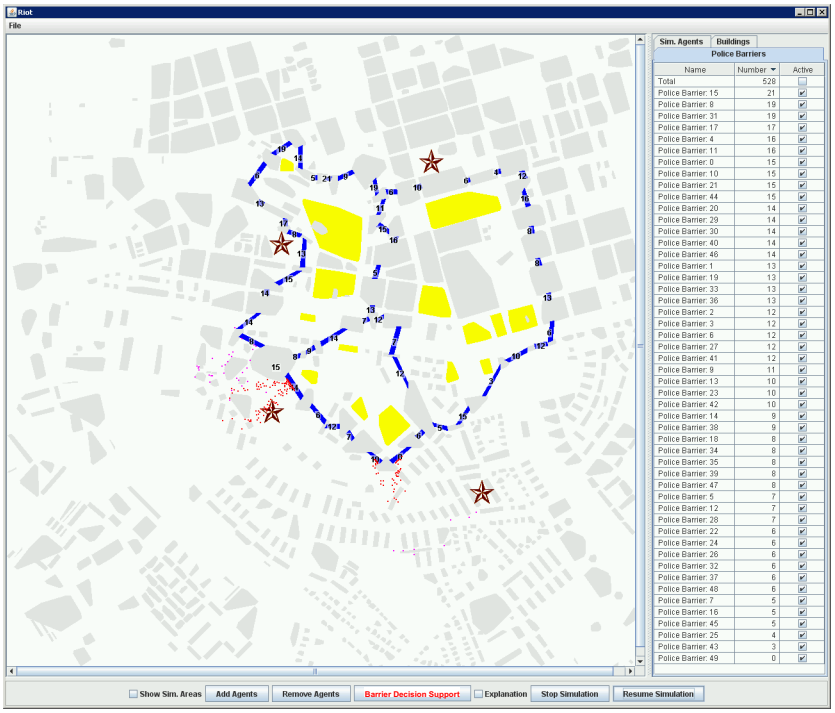

Figure 9. Graphical User Interface (GUI) of the Decision Support System.

The rioters are angry demonstrators. Demonstrators may turn into rioters when they are pushed together by the police. The demonstrators' priority is to assemble at the assembly points. The rioters' first priority is to attack the key buildings and then the police barriers.

The buttons at the bottom enables the user to show the simulated areas (the **Show Sim. Areas** checkbox). That is where the user may position his agents, using the **Add Agents** button. By selecting the **Explanation** checkbox, the user is informed of which simulations were used, and

their alpha values, for calculating the strength of the proposed police barriers. The **Start Simulation** button is used to start the simulator and thereby verifying that the proposed police barriers will hold.

The simulation may be paused and the system may provide a new barrier instantiation by pressing the **Barrier Decision Support** button.

When the **Police Barriers** tab is selected in the right field, all police barriers are listed in descending strength order. The police barriers can be selected and highlighted in both the list and in the map. If the mouse is held over a barrier in the map, a textbox appears showing its id and strength. The **Buildings** tab lists all buildings and their strength in descending strength order. The buildings can be selected and highlighted in both the list and in the map. If the mouse is held over a building in the map, a textbox appears showing its id and strength. The **Sim. Agents** tab lists all simulations when the **Show Sim. Areas** checkbox is selected and the simulations used to instantiate the barriers when the **Explanation** checkbox is selected.

Learning Strategies

The demonstrator consists of three different parts, a simulator called Riot Simulator Environment (*RSE*) developed in Java, a Scenario Production Module (*SPM*) developed in Matlab, and a Decision Support Module (*DSM*) also developed in Java.

The *RSE* is the core module in the demonstrator. Its functionality is used by all the other modules.

RSE supplies a Scenario Editor (Figure 10) with which it is possible to modify an environment by adding or removing buildings, or by adding or remove police barriers and agents.

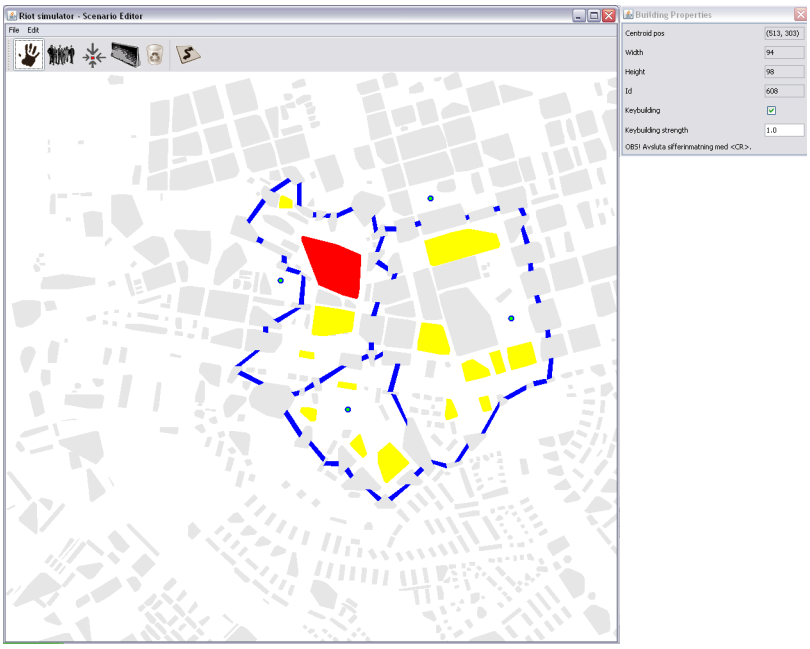

Figure 10. The Scenario Editor in *RSE* (red building chosen for editing).

The architecture of *RSE*, *DSM* and *SPM* is shown at a high abstraction level in Figure 11, Figure 12, and Figure 13. A new component in *RSE* is the agent generator, its main task is to create groups of agents and randomly place them into the simulation environment according to the zones illustrated in Figure 14.

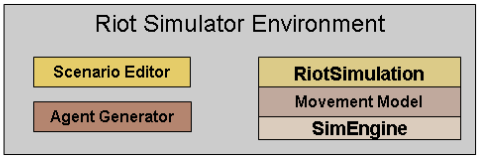

Figure 11. Main components in Riot Simulator Environment.

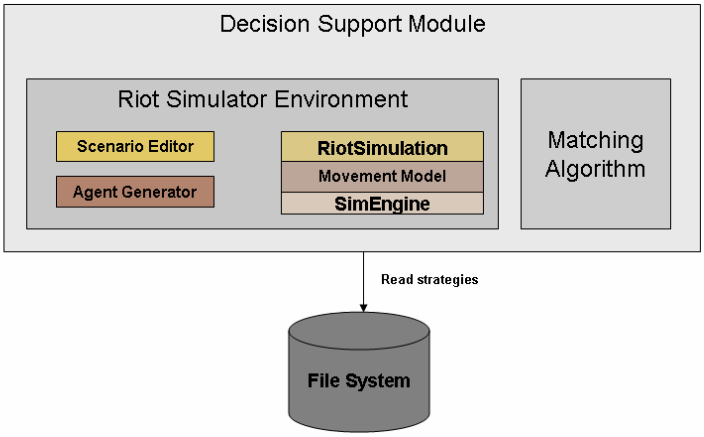

Figure 12. Decision Support Module (*DSM*).

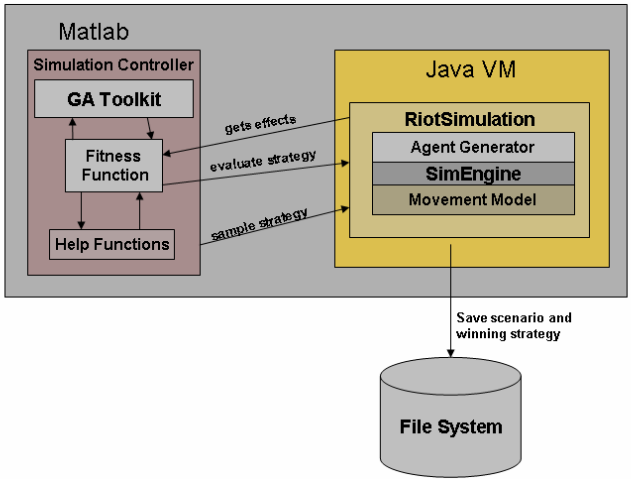

Figure 13. The interactions between *SPM* and *RSE*.

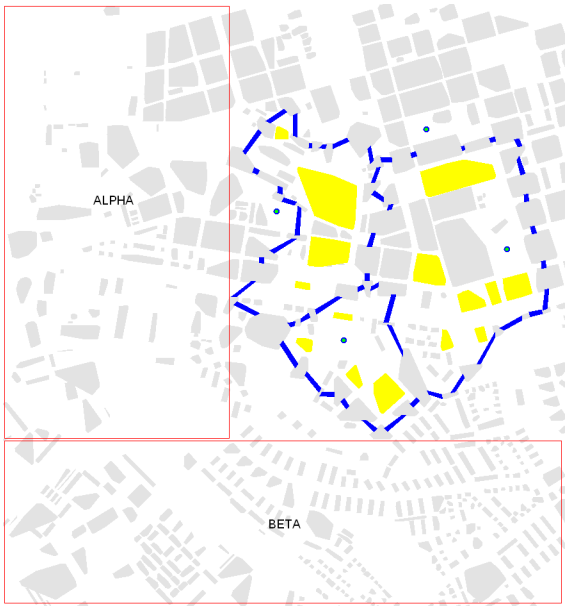

Figure 14. The two zones where the agent groups can be positioned.

Scenarios

The workflow for the production of scenarios is as follows:

- Create a basic scenario with the Scenario Editor,
- Run the basic scenario under the *SPM*,
- Generate agent groups at a random positions,
- Sample the best strategy for the ongoing scenario.

A scenario consists of static items, like police barriers, ordinary buildings, key buildings, attraction points, and dynamic items like randomly positioned groups of agents. The buildings are part of the environment of the scenario. The barriers and the agents are items created to build up a specific scenario.

In our experiment, we had a scenario with totally 739 buildings of which 15 are key buildings, 49 police barriers, 4 attraction points and 400 agents. The need for police resources in the above-mentioned scenario is proportional to the number N of violent agents. During the scenario production phase we assumed that the real need of police resources should be $R \geq 3N$. Roughly said the police resources engaged in a scenario must be 3 times more than the violent agents. In a scenario with 400 agents of which at least 70% are violent and the rest of them are peaceful or inclined to violence just in case of violent repression the real need of police means will be: $R = 3 * 400 * 0.7 = 840$.

To study an optimization problem, the number of resources employed at all police barriers must be lower than the real need. The total strength of police barriers was set to 700, 140 resources fewer than the assumed minimum. This constraint is important to make sense of the use of the genetic algorithm, so we can get the best partition of resources based on the smallest damage of key buildings. The problem at hand is to optimize the allocation of a scarce resource and to solve that we used Matlab's Genetic Algorithm Toolkit (*GA*). *GA* solves optimization problems by

implementing the principles of biological evolution, repeatedly modifying a population of individual points using rules modeled on gene combinations in biological reproduction. Due to its random nature, *GA* improves the chances of finding a global solution.

The decision support application uses a library of scenarios. The production of the library was achieved by variations of a basic scenario and the result of evolutionary variation of the police barrier strategy. The variations of the basic scenario are achieved creating different start positions of agent groups. The positions are chosen randomly from two zones called alpha and beta, Figure 14. Each group is composed of 4 agents. Totally there are 100 groups that have to be randomized over a specific area.

Learning strategies

The variation of police barrier strategy was achieved through the evolutionary methodology of *GA*. A police barrier was represented as a 5-bits array, a set of 49 arrays correspond to a complete strategy or a chromosome in *GA*, Figure 15.

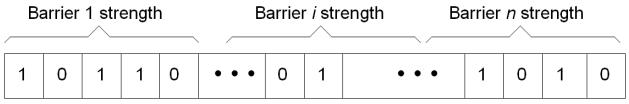

Figure 15. A binary representation (chromosome) of a complete strategy for all barrier strengths.

Each strategy can also be seen as an individual. In the problem of optimizing police barrier strengths and positions each individual corresponds to the collected information about the strength at every possible predetermined barrier position. Strength of zero at a certain barrier position corresponds to no barrier at this position. The higher the strength, the more agents a barrier can manage before it is broken and more resources are needed to man the barrier. The sum of all barrier strengths is constrained in the optimization. This corresponds in a real riot case to the situation that the resources for manning barriers are limited.

A group of individuals forms a population. The population of all individuals makes up the set of all alternative strategies for positioning and manning the barriers. The optimization of strategies for barrier positions and strengths is carried out in each generation by evaluating each strategy by itself. Each strategy is scored based on its success in the riot control simulation from two aspects, on the one hand in protecting certain designated buildings and on the other hand that the barriers themselves are not disrupted by the rioters.

An obvious balance is that the barriers must be placed in positions where they are at risk themselves in order to protect the designated buildings. However, they must not be placed in such a way that they are at risk needlessly nor be given such a low strength that they are easily broken without giving protection to the designated buildings. A suboptimal solution will be achieved if proper consideration is not taken to the fact that both barriers and designated buildings may become destroyed by the rioters in the simulation.

The initial set of strategies is generated randomly. The start population for this experiment was composed of 20 randomly generated individuals (or chromosomes). The progenies were created according to the principles of biological evolution in *GA*. As this set of alternative strategies develops, good strategies with high scores are chosen and generate offspring. This is done in such

a way that all strategies in one generation are evaluated by the riot control simulator and awarded score points from the perspective of how well the designated buildings were spared and that the barriers themselves where not disrupted by the rioters. From this scoring all strategies a rank based selection of individuals is made. The rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness *N* (number of chromosomes in population), in our case 20.

To find the best strategy for a scenario, it was necessary to run this scenario several times. *GA* simulated, 20 * *X* times each scenario to find the best strategy, where *X* depends on the number of generations needed before convergence was manifested, $X = \{1, ..., 100\}$. In total we performed 129332 simulations, with a computation time for each scenario of circa 15 minutes. Each strategy was evaluated by RiotSimulation. When the results converge GA returns all the results to a controller function that selects the best strategy from the current run. The best strategy was sampled by running the **RiotSimulation** in sampling mode. We produced a library of 150 scenarios. Each scenario is saved in its own file.

In Figure 16 we observe the scores of 129332 simulations. A perfect minimal score of 18350 is found 48.52% of the time. Each additional 50 points correspond to one destroyed building. Smaller fluctuations correspond to broken police barriers. 118547 strategies are quite successful. These are used for decision support, while the remaining 10785 with the higher scores are discarded.

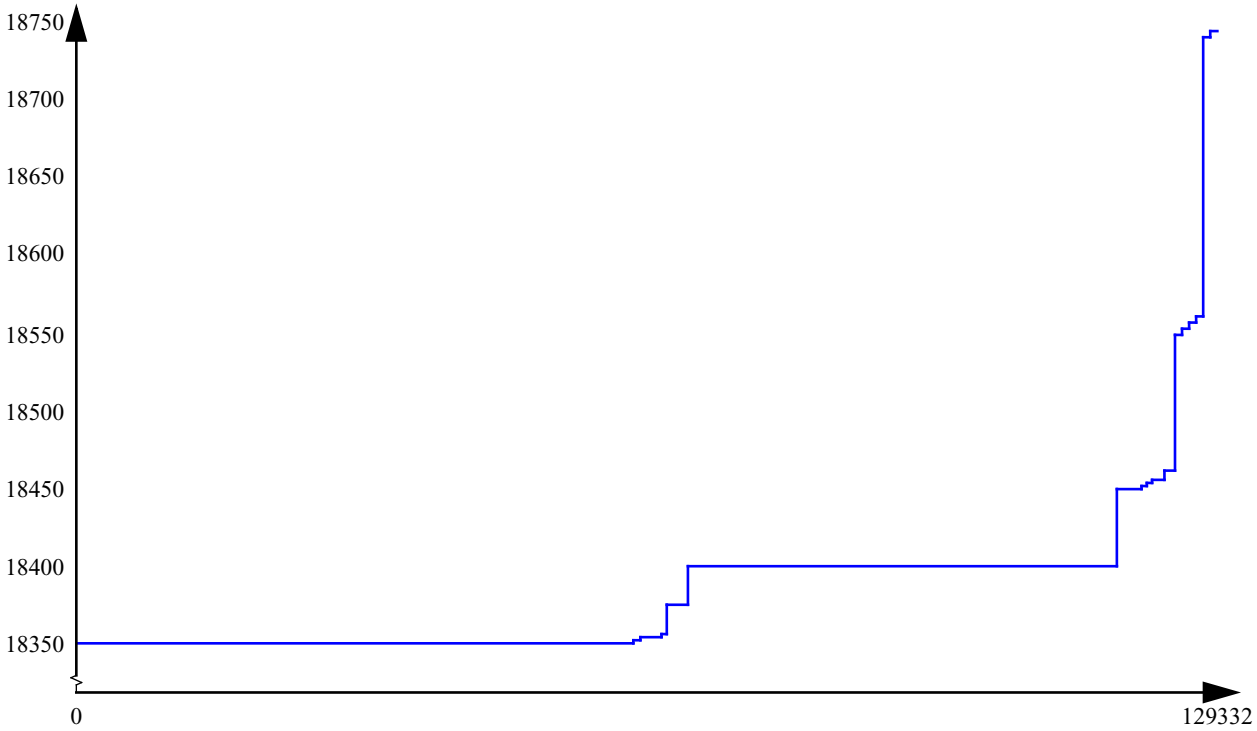

Figure 16. Scores of 129332 groups of agents.

Correction inserted

Stochastic Agent-based Simulation

In this section we describe the simulation that is used to score strategies in the genetic algorithm.

Stochastic agent-based simulation (SABS) is based on stochastic agent models, embedded simulations and the predicted effects that are the output of the simulation. An agent is anything that can perceive using its sensors and act using its effectors. An agent can be an individual or a group of individuals (Russell & Norvig 1994). In our simulation, a software agent is a representation of a real world agent.

Inputs to the simulation are strategies, uncertainty-based representation of the situation, and behavior. Strategies are represented here by all barrier positions and strengths. Situations are represented by positions of real world agents, both rioters and peaceful protesters, with an uncertainty-based estimate of their hostility. By specifying the number in a range from zero to one the user may enter its own belief of the median value of the statistical distribution of hostility. Simulation runs until a pre-defined stop time, determining how long the situation is predicted.

Scenarios have to be developed for each area of interest. They are not assumed to be generic in nature. The riot simulator uses a simple 2D geographical model of a 2000 x 2000 meter part of a city. This city has been extensively 3D-modelled using data from airborne laser radar. The XYZ 3D models of buildings on the city ground surface, with varying Z elevation, have been projected down to the XY (Z = 0) 2D plane, and the convex hull determined for each building 2D projection. These hulls have been used as the representation of buildings (obstacles), in the simulation, in total about 739 buildings. Using convex hulls allows the use of simpler and faster search algorithms for the agents trying to find their way towards their goals.

Given a specific scenario, barrier strategies and protesters actions, embedded simulation can be used. Since the simulations use stochastic agents, different effects may occur from one simulation to the next. Simulated effects are destroyed buildings of importance and destruction of the barriers. The prioritization between buildings and barriers is presumed to be made by a tactical commander. In learning strategies by genetic algorithms and simulation we put a weight on how a destroyed building is prioritized compared to a disrupted barrier.

The agents can be simulated serially in a single thread or in parallel as individual threads. At the simulation start, they (typically 400 agents in total) are distributed at random in small groups in an area well outside the region of their goals which are situated closer to the city centre, most of them inside police barriers. They are initiated, each with different aggression levels, which are crucial for their choice of goals, or attractors. If agents are congested densely together, as in narrow bottleneck passages between buildings or onto a police barrier, their aggression level increases.

In Figure 17 we observe a scenario where police barriers are broken through. Active police barriers are blue. Ordinary buildings are grey. Undamaged key buildings are yellow. Attraction points are green with a blue circle around. Demonstrators and rioters here have approached from the south-west. More peaceful demonstrators (black and pink dots) want to gather at an area (attraction point) up to the left. Angry rioters (red dots) have split up in three groups, that has broken through four barriers (dark grey) and, so far, has damaged five key buildings (also dark grey).

Figure 17. A snapshot of a riot simulation.

The motion model of the agents is similar of that in (Helbing & Molnár 1995). It is driven by different attractive forces. There are also repulsive forces. The attracting objects in our simulation are singular attraction points, key buildings and police barriers. The attraction forces of these (here called *global attractors*) are linearly declining with distance as following, and are computed individually for each agent:

$$\begin{aligned} \text{Attraction\_point\_attr\_force} &= 1 - \frac{\text{distance}}{1500}, \\ \text{Key\_building\_attr\_force} &= 1 - \frac{\text{distance}}{1000}, \\ \text{Police\_barrier\_attr\_force} &= 1 - \frac{\text{distance}}{500}. \end{aligned} \tag{14}$$

All forces are set to zero if negative, but the agents are always placed close enough to the attractors to feel a non-zero attraction force. For an agent with low aggression level (< 0.6), the attraction point (if several are present) giving the largest attraction force is chosen as global attractor for that agent. These agents correspond to people that only want to gather at a certain place to engage in a peaceful manifestation. For an aggressive agent, the global attractor is chosen as that key building or police barrier among all of those which have the strongest attraction force on that agent. These agents correspond to aggressive rioters that want to attack buildings or the police. A key building or a police barrier is broken when the sum of the aggression of the agents

attacking it exceeds a preset strength for that object. Destroyed key buildings[1] or broken police barriers lose their attraction force, which means that the focus of aggressive agents attacking key buildings or police barriers will change to another attractor once the object they attack is damaged enough (broken). A broken police barrier no longer constitutes an active barrier, so all agents can pass through it without delay. If the object withstands an ongoing attack, the agents (in the current simulator version) will keep on attacking the object for ever, and not move away from it. This is also the case for attraction points for peaceful agents; they keep on moving around in its vicinity without attacking any objects.

The global attractor chosen for an agent, causes the agent to find a suitable path among the buildings on its way towards that attractor. Agents keep a local awareness about the closest buildings that may obstruct their path. The local awareness about buildings is updated within a circular region with constant radius (typically 100 meters) every time an agent comes close to the perimeter of the last updated awareness region. This strongly reduces the amount of buildings that have to be considered when finding the path locally, reducing computation time. The search algorithm does not try to find the shortest path all the way to the current global attractor (like an A* search (Russel & Norvig 1994)); rather it always looks more locally and finds the corner of the closest obstructing building that has to be bypassed. The closest obstructing building may not necessarily be the closest building on the straight path towards the global attractor; the choice might be that a more distant building is the first that has to be passed if a straight line can be found towards one of its corners. This is found using a step-wise search-forward with backtracking path-straightening-out mechanism that, when passing a corner, tries to find the best next corner on the path while forecasting up to five corner points beyond that. On their way to the global attractors, the corners of the buildings that have to be bypassed act as local attractors that are updated with new corners after reaching them. This process seldom gives the shortest path to the global attractor, but it probably better coincides with the path chosen by a stressed person trying to find his / her way locally towards a more distant, still invisible, goal. Furthermore, the paths towards the local attractors are most often not straight lines between corners, the social forces between agents' result in repulsive displacements of them; the larger, the higher the density of agents, as explained below.

No complex collective group behavior is implemented for the agents so far; they interact only with a repulsive *social force* according to:

$$\text{Social\_repuls\_force} = e^{(2 \cdot \text{agent\_radius} - \text{distance\_between\_agents})}, \tag{15}$$

where agent_radius is typically 0.30 meters.

This force results in a dispersion of the agents. The agents only feel a local awareness of their neighboring agents: To reduce computation load, the 2D simulation region is divided into 10 x 10 meters squares, and only the agents residing in the eight squares around an agents' square are

1. Destruction of a key building is not to be interpreted as if the building has been torn down brick by brick by rioters, but rather that enough rioters were able to reach it at all (maybe a consequence of a bad police barrier strategy), and then do whatever damage to it, or disturb the activity going on in it.

considered for social force computation for that agent. The resulting force on the agent is the added force from all neighboring agents.

In (Helbing and Molnár 1995), there is also an inertial force representing the current motional state of the agent. It is not modeled here, since the agents are massless. However, the social force actually gives an effect similar to it; in a large enough group of agents, something like a collective viscous flow of the agents is observed, as a consequence of their social interactions, effectively giving a weak inertia to the group as such.

After summing the force vector of the local attractor with the resulting sum of social forces from neighboring agents, a resulting force is obtained for an agent. It is normalized to only keep the direction of movement, and is multiplied with the *average walking speed* of the agent to get the walking velocity. The walking speed of the agents is uniformly distributed from agent to agent between 0.6 and 1.6 units per second. This walking speed distribution contributes to an increasing geographical spread in the direction of movement of the agents when they are moving in larger areas without obstructing buildings.

Now, the agent moves according to the walking velocity for the next simulation step. If it then moves into an unbroken police barrier, the move is not done. If it moves into a building (normally due to an added displacement caused by the social force), the move is replaced with the vector along the building edge which points in the direction of the local attractor, often a corner of the same building according to the discussion of local attractors above. Finally, all buildings actually repel agents weakly according to an ad hoc chosen formula that apply only if the agent is closer than 4 meters from the building:

$$\text{Building\_repuls\_force} = e^{-\text{distance}} + 1\,, \tag{16}$$

which reduces the risk that agents come too close to a building, but they can still bump into it if the social forces are strong enough.

A typical simulation starts with the demonstrators and rioters gathered in a large region well outside the central town. Typically 1/3 of them are usually peaceful demonstrators, the rest are rioters. They start to move towards their respective attractors. The demonstrators gather around attraction points and the rioters attack police barriers. Depending on the strength of a barrier, a varying amount of rioters is needed to break the barrier. When broken, the rioters enter the central town with the intent to damage key buildings. When a key building is damaged enough, another key building will gain their attention. Sometimes the rioters break up in two or more groups, focusing on different key buildings. This behaviour is more or less random, it can happen for a larger group that reaches a building, which is then forced to split up and move along the two different sides of the building. After the passage, they might be attracted by different attractors due to the separate positions of the two groups.

Conclusions

We have demonstrated that it is possible to derive riot control strategies using genetic algorithms and stochastic agent-based simulation. We have developed a Decision Support System using a decision making algorithm where a current situation is compared to all simulated situations. A linear combination of control strategies whose corresponding weighted superposition of simulated situations most closely resembles the current situation, is given as decision support.

References

Dixon, D. S., and W. N. Reynolds. 2003. The BASP Agent-Based Modeling Framework: Applications, Scenarios and Lessons Learned. Paper presented at the 36th Hawaii International Conference on System Sciences, January 6–9, in Waikoloa, Hawaii.

Graves, T., R. Picard, and S. Upton. 2000. Improving Rule Bases for Agent Based Simulations. Unclassified Report 00-2566. Los Alamos: Los Alamos National Laboratory.

Grieger, D. 2003. An Overview of Crowd Control Theory and Considerations for the Employment of Non-Lethal Weapons. DSTO-GD-0373. Edinburgh, Australia: Defence Science and Technology Organisation.

Helbing, Dirk, and Peter Molnár. 1995. Social force model for pedestrian dynamics. In *Physical Review E* 51(5): 4282–4286.

Helbing, Dirk, Illés Farkas, and Tamás Vicsek. 2000. Simulating dynamical features of escape panic. In *Nature* 407(6803): 487–490.

Holland, J. H. 1973. Genetic Algorithms and the Optimal Allocation of Trials. In *SIAM Journal on Computing* 2(2): 88–105.

Project Albert. 2008. [Online]. Available: http://www.projectalbert.org

Russell, S. J., and P. Norvig. 1994. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs: Prentice-Hall.

Schubert, Johan, and Robert Suzić. 2007. Decision support for crowd control: using genetic algorithms with simulation to learn control strategies. Paper presented at the Third IEEE Workshop on Situation Management, October 30, in Orlando, FL.

Wolfe, Philip. 1959. The Simplex Method for Quadratic Programming. In *Econometrica* 27(3): 382–398.