Intelligent task scheduling in sensor networks

Introducing three new scheduling methodologies

LTZE3 W.L. van Norden

April 2005

Delft University of Technology

Media and Knowledge Engineering
Man-Machine Interaction group

Delft Cooperation on Intelligent Systems

Royal Netherlands Naval College

Combat System Department

The Principle Concerning Multifunctional Devices:

The fewer functions any device is required to perform, the more perfectly it can perform those functions.

Patton's Law:

A good plan today is better than a perfect plan tomorrow.

Optimum Optimorum Principle:

There comes a time when one must stop suggesting and evaluating new solutions, and get on with the job of analyzing and finally implementing one pretty good solution.

Graduation Committee

drs. dr.	L.J.M. Rothkrantz
dr. ir.	C.A.P.G. van der Mast
ir.	F. Ververs
KLTZE ir.	F. Bolderheij
drs.	J.L. de Jong

Abstract

Ever more complex sensors have become available to maintain situational awareness during missions. Each has different capabilities and is therefore suited to one or more sensor functions. Choosing the best suited sensor for any sensor function is based on sensor capabilities as well as task attributes. In highly dynamic environments these characteristics can change rapidly, leading to a shift in sensor allocation. To increase performance of the entire sensor network the total set of sensors should be scheduled in a single system. This thesis puts forward and compares three new methods for scheduling prioritised tasks in a sensor network. The first scheduler is based on fuzzy Lyapunov synthesis. This scheduler uses a different buffer for each type of sensor function in which incoming task requests are placed. Whenever a sensor is available to execute a new task, a task is chosen from the largest buffer based on the sensors capabilities. This leads to a fast scheduling procedure with good performance. The second scheduler uses a genetic algorithm (GA) off-line to determine the optimal set of schedules for all sensors. Based on these optimal schedules a neural network (NN) is (re)trained to be used on-line. The use of the NN leads to a very fast scheduler, but one that guarantees neither optimality nor transparency. The third approach is a novel on-line use of a GA. This scheduler uses the GA to optimise the set of schedules, this time in a hybrid form. The solution of the scheduler currently used in sensor scheduling is included in the initial population; therefore the GA always improves the set of schedules. Using this implementation of a GA leads to a system that keeps optimising by re-scheduling. Tests showed that this novel on-line use of GA leads to a robust scheduler with highest performance.

Preface

This thesis research was conducted at the Delft Cooperation on Intelligent Systems (DECIS) laboratory. This lab is supported by Thales Research and Technology Netherlands, Applied Physics Laboratory (TNO), Delft University of Technology and the University of Amsterdam.

The new approach to the C2 process was developed as part of the STATOR research program supported by Thales Naval Netherlands, the Royal Netherlands Naval College and the International Research Centre for Telecommunicationstransmission and Radar of the Delft University of Technology.

Acknowledgements

I'd like to thank my supervisors drs. dr. L.J.M. Rothkrantz of the Delft University of Technology, KLTZE ir. F. Bolderheij of the Royal Netherlands Naval College and drs. J.L. de Jong of Thales Research and Technology at the Delft Cooperation on Intelligent Systems (DECIS) laboratory for their much needed guidance and support during my thesis research.

Wilbert van Norden

Delft, April 2005

Table of contents

Abstract iii

Preface v

Table of contents vii

List of figures xi

Abbreviations & acronyms xiii

1 Introduction

1.1 Problem description 1

1.2 Relevance 2

1.3 Goals 3

1.4 Outline 3

2 The radar scheduling problem

2.1 The general RSP 5

2.2 Scheduling in sensor networks 7

2.3 Task duration 8

2.4 Summary 9

3 Current radar schedulers

3.1 Grouping of tasks for scheduling 11

3.2 Scheduling based on matrix representation of the sky 13

3.3 Queuing of dwell requests 16

3.4 Scheduling with intelligent agents 18

3.5 Cons of current techniques 18

3.6 Summary 20

4 Overview of applicable scheduling techniques

4.1 Fuzzy Lyapunov synthesis for scheduling 21
4.2 Scheduling with gatekeepers 23
4.3 Neural network scheduling 24
4.4 Scheduling with genetic algorithms 24
4.5 Constraint satisfaction programming in scheduling 26
4.6 Sequence scheduling 27
4.7 Multiple families of jobs in the JSSP 31
4.8 Choosing a proper scheduling technique 32

5 Modelling the schedulers

5.1 Overview of the entire system 37
5.2 Data flow in sensor scheduling 40
5.3 Fuzzy Lyapunov based scheduler 42
5.4 Genetic neural network 47
5.5 Online use of a GA 53
5.6 Hybridisation 54
5.7 Evaluation of chosen models 55

6 Implementation

6.1 Modelling of tasks 57
6.2 Fuzzy Lyapunov based scheduler 58
6.3 Genetic neural network scheduler 62
6.4 Online use of GA 67
6.5 Hybrid scheduling 68
6.6 Evaluation tools 68

7 Results

7.1 Simulation 71

7.2 Task requests 72

7.3 First-in-first-out 74

7.4 Results of the three developed schedulers 76

7.5 Comparison 81

7.6 Improvements through hybridisation 82

8 Conclusions & Recommendations

8.1 Conclusions 87

8.2 Recommendations 88

References 89

Appendices

A. Theoretical Background 91

B. Table with simulation objects 95

C. Results of hybrid scheduler 99

List of figures

2.1 Left: Multiple SFR architecture. Right: Single MFR architecture 5

2.2 Different types of scheduling in a single MFR 7

3.1 Outline of dwell scheduler proposed by Huizing and Bloemen, taken from [8] 17

4.1 Ten tasks that need to be scheduled based on their ready and due dates 29

4.2 Construction of super sequence for problem illustrated in figure 4.1 29

4.3 Algorithm to construct master sequences 30

5.1 Sensor control loop 37

5.2 NCW sensor loop 39

5.3 The context diagram for sensor scheduling 40

5.4 Data flow for task generation in sensor scheduling 71

5.5 Outline of a fuzzy Lyapunov radar scheduler 44

5.6 The four data processes in Lyapunov based scheduling 46

5.7 Outline of a scheduler using a NN and a GA to update it 47

5.8 igmoid function: $y = f(x) = \frac{1}{1+e^{-x}}$ 52

6.1 Functions and variables in the Lyapunov based scheduler 59

6.2 Utility of the schedule for 700 random task requests 60

6.3 Number of dropped tasks in scheduling 700 random tasks 60
6.4 Utility for 700 random task requests for small γ 61
6.5 Utility for 700 random tasks for different values of k 62
6.6 Software architecture of the Genetic-Neural Network 63
6.7 Matlab code for 'swapping' function 64
6.8 Influence of α on the utility of the schedule 64
6.9 Generations needed for maximum fitness for chromosome length 10 over ten runs 65
6.10 Maximum fitness after 500 generations over ten runs for chromosome length 10 65
6.11 Error development in scheduling intervals 66
6.12 Error development in the scheduling advisor 66
6.13 Program flow chart of sequential implemented on-line use of GA in scheduling 67
6.14 Function for the chromosome mapping in a multi sensor situation 69
6.15 Reverse swapping function 70

7.1 Code to make task requests out of column two of the object-table 73
7.2 Histogram of the requested tasks during simulation 74
7.3 Utility in time using the FIFO scheduler 75
7.4 Utility of Lyapunov scheduler 76
7.5 Utility of GNN scheduler 76
7.6 Utility of online GA scheduler 77
7.7 Dropped tasks with Lyapunov scheduling 78
7.8 Dropped tasks with GNN scheduling 78
7.9 Dropped tasks with online GA scheduling 78
7.10 Sensor use with Lyapunov scheduling 80
7.11 Sensor use with GNN scheduling 80
7.12 Sensor use with online GA scheduling 80
7.13 Utilities of the Lyapunov, the online GA and the hybrid schedulers 83
7.14 Comparing the online GA scheduler and the hybrid scheduler with the fuzzy Lyapunov based scheduler, N = 50 and the maximum number of generations is 5 84

Abbreviations & Acronyms

APAR	Active Phased Array Radar
CAF	Clear-A-Fraction
CLB	Clear Largest Buffer
CMS	Combat Management System
CP	Clearing Policy
FIFO	First In First Out
GA	Genetic Algorithm
GNN	Genetic Neural Network
HS	Horizon Search
JSSP	Job-Shop Scheduling Problem
LVS	Limited Volume Search
MAS	Multi Agent System
MFR	Multi Function Radar
MG	Midcourse Guidance
NCW	Network Centric Warfare
NN	Neural Network
OODA	Observe-Orient-Decide-Act
PA	Phased Array
RSP	Radar Scheduling Problem
SFR	Single Function Radar
STATOR	Sensor Tuning And Timing on Object Request
THP	Tracking High Priority
TI	Terminal Illumination
TLP	Tracking Low Priority
TWS	Track While Scan
VS	Volume Search

1 Introduction

This chapter introduces the subject of this thesis research. Section 1.1 states the problem definition. The relevance of this problem is given in section 1.2. The goals of this thesis are stated in section 1.3. In the final section, 1.4, the outline of this thesis report is given.

1.1 Problem description

Ever more complex sensors have become available to create and maintain situational awareness during missions. Each has different capabilities and is therefore suited to one or more functions. Choosing the best suited sensor for any sensor function is based on sensor capabilities as well as task attributes. In highly dynamic environments these characteristics can change rapidly, leading to a shift in sensor allocation. To increase performance of the entire sensor network the total set of sensors should be scheduled as a single system.

When a task request is allocated to a sensor it needs to be scheduled in the best possible way. The problem of scheduling tasks in a single sensor is known as the radar scheduling problem. Combining scheduling solutions for this single sensor problem with the problem of allocating a task to a sensor would most likely improve performance further.

This thesis gives an overview of current scheduling techniques in sensors. Schedulers in other domains are also evaluated on their usability in sensor scheduling. The most promising schedulers will be implemented and tested to see if they are usable in the sensor domain. The implementation is based on the command and control situation as introduced by Bolderheij [3].

1.2 Relevance

On the new Air Defence and Command Frigates of the Royal Netherlands Navy the APAR (Active Phased Array Radar) is used for air defence. This is a Multi Function Radar (MFR), meaning that the system can perform multiple sensor functions like search, track and weapon guidance. Since these different sensor functions can now be performed by a single sensor the underlying command structure is more complex compared to the structure for task specific sensors.

In search for optimisation of the performance of APAR the Royal Netherlands Naval College along with Thales Netherlands and the International Research Centre for Telecommunicationstransmission and Radar of the Delft University of Technology started a project called STATOR (Sensor Tuning And Timing on Object Request). The goal of this project is to set the control parameters of a MFR with operational knowledge and thus optimising performance. Ultimately the goal is to develop a system to propose sensor control parameters for a set of sensors, e.g. in a taskforce, following up on the increasing interest in Network Centric Warfare (NCW).

When looking for ways to improve the performance of a sensor network, it is obvious to look to the scheduling algorithm that decides which task will be performed when. For a single sensor this scheduling problem is known as the Radar Scheduling Problem (RSP). Another influence on performance is the choice of sensor. Optimising the matching between sensor and task will lead to better performance of the entire system and thus maximise survivability in military missions.

1.3 Goals

Improving the performance of a sensor network is a complex problem. The focus will therefore lie on testing and implementing promising techniques that are already available. The goals can be stated as:

1. Define the problem domain and identify the required characteristics in the scheduler;
2. Evaluate current sensor schedulers based on the stated requirements;
3. Evaluate schedulers from other domains on their applicability in the sensor domain;
4. Test and implement promising techniques;
5. Evaluate and compare these techniques based on their performance and applicability.

1.4 Outline

Chapter two will discuss the general RSP and the problem in scheduling sensor networks. Some current solutions to the general problem are discussed in chapter three. In chapter four general approaches to solving scheduling problems are introduced and discussed. Based on these techniques three models for solving the RSP are put forward in chapter five. This chapter also gives an outline of the view on sensor management as it is used in modelling and implementing the schedulers. Chapter six discusses the implementation details of the schedulers. The proposed scheduling systems are tested in a simulated environment. Both this environment and test results are given in chapter seven. Conclusions and recommendations are finally stated in chapter eight.

2 The radar scheduling problem

This chapter introduces the scheduling problem in sensor systems. Section 2.1 will first discuss the general radar scheduling problem for MFR. This problem is then expanded in section 2.2 to the scheduling problem in sensor networks. Finally, section 2.3 will give some information on the duration of several radar functions.

2.1 The general RSP

The primary goal of the entire set of radar systems aboard a naval vessel is to create and maintain situational awareness. Achieving this goal generally means that two types of radar functions are to be performed by the overall system: search and track. Weapon guidance is a third function type that results from weapon deployment to minimise risk to the mission. Requests to perform any of these three task types are generally made by the combat management system (CMS). The use of several single function radars (SFRs) results in a relatively simple scheduling problem. All tasks are sent to the appropriate SFR. Problems that occur with such a configuration are: radar beam interference between different radars, idle times for specific SFRs and more sensors on board means that the stability of the ship is influenced. A solution to this problem is the use of phased array (PA) multi function radar (MFR). This radar doesn't rotate due the electronic beamsteering and is able to perform several functions. Beam steering has the advantage that direction of the radar face is unnecessary. This however leads to the problem of scheduling the tasks within the system as shown in figure 2.1.

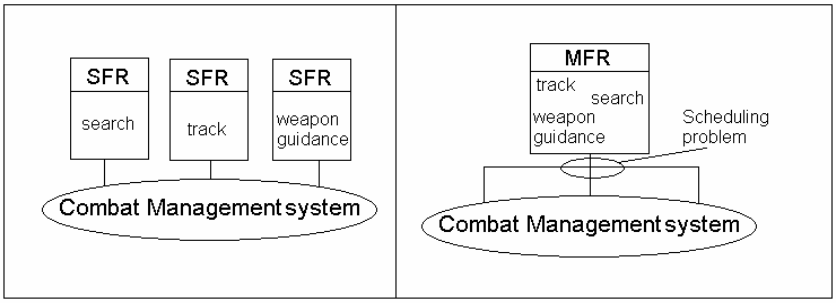

Figure 2.1 Left: Multiple SFR architecture. Right: Single MFR architecture

The basic structure of a task consists of temporal, directional and emission demands. The first states how often a radar dwell has to be done within a time interval. The second states in which direction it should be sent and states the azimuth and elevation beam width. The last gives the parameter for a single radar dwell, e.g. carrier frequency, the number of radar bursts in a dwell and the number of pulses within a single burst.

The MFR considered in this thesis is able to perform all types of sensor functions mentioned earlier. Within sensor systems however, a subdivision might be necessary in weapon guidance tasks due to the different stages of weapons guidance and their particular needs. This sub-division leads to an approach where four task types are identified: search, track, midcourse guidance (MG) and terminal illumination (TI). The latter is considered a separate type due to the difference in parameters between MG of a missile and the terminal guidance phase. In terminal guidance the emitted power by the radar is far greater than with the other task types; forcing the system to introduce set-up times when switching between TI and another type of sensor function. Note this does not hold for all systems, more specialised sensors (like SFR) do not have this property. Set-up times are introduced here to include the principle in the conceptual design of the scheduler.

All these demands affect the complexity of the scheduler. Another important factor is that the MFR under consideration can be rotating or fixed. Rotation means that the observed area is not always in reach, implying more temporal demands on scheduling. Fixed radar has the entire area in constant reach leading to more freedom in scheduling. Using this freedom well means an increase in performance of the overall system. The algorithms used for scheduling however become more complex.

Two different scheduling problems can now be identified for the MFR. Firstly, the scheduling of tasks for a single MFR or a sensor network and secondly, scheduling radar dwells based on the tasks within a sensor. Scheduling algorithms must be placed within this context in order to understand their usefulness for generalisation to multi sensor networks where task allocation is done centralised and dwell scheduling decentralised within the sensor. Since the latter is done within the sensor the focus in this thesis is task scheduling. This principle is illustrated in figure 2.2 for a single MFR.

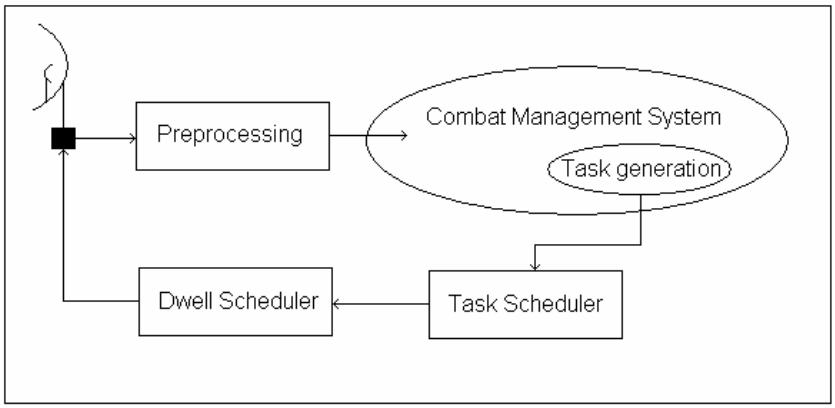

Figure 2.2 Different types of scheduling in a single MFR

2.2 Scheduling in sensor networks

In the last years more and more research is done in the field of NCW. The subdivision of task and dwell scheduling is an interesting notion for combining sensor capabilities. The generation of tasks is usually done by the CMS on each ship separately, since the dwells are scheduled per sensor as well. Now, these tasks can be generated by a one system that can assign tasks to all available sensors while it need not schedule the actual dwells for each of those tasks. The advantage of this approach is that tasks can be switched between sensors rapidly without extensive communication between the sensors. A problem that is now introduced is the allocation of a sensor to a task. In terms of range this is easily solved, a task can only be assigned to a sensor if the sensor is in range.

Other considerations however are more complex. The accuracy of the sensor for instance is an important measure. Another factor is the detection probability. If a threat is expected in a certain area the choice would be to use the sensor with the best trade off between accuracy, detection probability and task duration. The latter is of importance because if a sensor probes the area more accurately, it will need more time to execute the tasks and a single sensor can only perform one task at a time.

This leads to the next problem in managing the sensor network, the number of tasks assigned to each sensor. Based on the previous the choice would be to use the best sensor for each task. This however could be impossible in time when there are too many tasks to perform. Less suited sensors should then be used for less important tasks. An example could be to use optical sensors for tracking when other sensors are busy with weapon guidance.

Most scheduling schemes used today are based on separate algorithms for task allocation to a sensor and scheduling tasks within a sensor. This however does not necessarily lead to the most optimal use of all sensors in the network. Integration of both the allocation and the scheduling is therefore proposed in this thesis to further optimise the performance of the sensor network. Sensor characteristics are important for the network scheduling and tasks should be distributed as evenly as possible over the available sensors. Changing the task allocation from one sensor to another can now be done rapidly which complies with the real-time demands in the military domain.

2.3 Task duration

In section 2.2 the task duration was identified in a trade off measure in sensor allocation. Therefore it is necessary to get an indication about the length of radar functions. Also, an indication is needed about task length when looking for possible scheduling solutions. The scheduling of tasks that take up a days work has a very different demand on computation time then scheduling e.g. computer tasks that take up several milliseconds.

To give a flavour of the length of tasks a single radar pulse is first considered. Radar emissions are electromagnetic waves and travel with the speed of light. Looking at an object at 40 km, a single burst would take up 0.3 ms, Eq. (2.1). The factor two is added since the electromagnetic wave has to travel to the object and then back to the receiver.

$$t = \frac{2r}{c} = \frac{2 \cdot 40 \cdot 10^3}{3 \cdot 10^8} \approx 3 \cdot 10^{-4} s \qquad (2.1)$$

Assuming a beam width of 1° in both azimuth and elevation and two bursts for an emission, this is the absolute minimum of bursts. For a horizon search of 90° in azimuth this means that the task duration is at least $3 \cdot 10^{-4} \cdot 90 \cdot 2 = 54$ ms. Executing a limited volume search at the same distance in an area of 15° by 15° the task will take $3 \cdot 10^{-4} \cdot 15 \cdot 15 \cdot 2 = 135$ ms. Scheduling tasks with these short durations means that the scheduler has to be fast in order to keep up. Another factor is deciding which tasks to perform and which to drop.

2.4 Summary

The general RSP was introduced in this chapter by identifying the four main sensor functions executed by a single MFR. Determining which sensor function is executed when based on the tasks requested by the CMS is the core of the RSP. Dwell scheduling and task scheduling are separated so a set of sensors can be considered without looking at the details of dwell scheduling. This also leads to the concept of centralised task scheduling and decentralised dwell scheduling which could be the first step to NCW. Due to the dependency of task scheduling and sensor allocation there is a need to integrate these processes in a single scheduling scheme. Looking for schedulers means that an indication is needed about task durations, constraining the computational needs of scheduling algorithms. Since task durations in the sensor domain are in the order of milliseconds fast schedulers are needed.

3 Current radar schedulers

Especially after the development of the MFR many research was done in solving the RSP. During the years many ‘intelligent’ scheduling methodologies were introduced. This chapter introduces three of these schedulers in sections 3.1 – 3.3. Other solutions such as the countless heuristics that were developed are not discussed at this point. Section 3.4 describes the approach of a current research project in task scheduling.

3.1 Grouping of tasks for scheduling

In [2], Barbato and Giustiniani propose an algorithm for a rotating PA MFR. Within this radar they identify three basic tasks:

1. Horizon search (HS)	Scan to locate sea skimming missiles. Search is conducted in 360º azimuth and from sea level to BWE(0)º in elevation;
2. Volume search (VS)	Scan to locate air targets in 360º azimuth and from sea level to 70º elevation;
3. Tracking	Track tasks are sub-divided in four groups: track-while-scan (TWS); low priority (TLP); high priority (THP); Confirmation.

Each of these tasks has demands on the number of scans that need to be performed. The number of scans for HS is calculated based on the BWA (azimuth beamwidth) of the system. Numbering all beams (or dwells) necessary to fill 360º azimuth leads to division of odd and even beams performed in the odd and even scans. Doing so means that the entire HS task is conducted once every two scans.

The VS covers a great deal of space and the shape of the antenna is therefore adapted to minimise beam steering. The topside of the radarface is pointed at 35º elevation. To calculate the number of beams necessary for the entire volume the BWE (beamwidth in elevation) has to be known.

Calculation of the number of beams required for VS, is more complex than calculating it for HS because BWE and BWA are both functions of the elevation (e), as given in Eq. (3.1) and (3.2). The BWE at sealevel is denoted as BWE_{nom}, the number of beams in azimuth as NAZ and NAZ(0) is defined as 360º / BWA (equal to the total number of beams for HS).

$$BWE(e) = \frac{BWE_{nom}}{\cos(e - 35^{o})} \tag{3.1}$$

$$NAZ(e) = NAZ(0) \cdot \cos(e) \tag{3.2}$$

In Eq. (3.1) and (3.2) the variable e is a discrete value, the radar beams can be directed at fixed elevations. This is done in such a way that the outcome of the cosine is non-negative. The elevation of the topside of the radar face allows for different elevations to be searched with the same relative steering command. Many ways exist to divide the space, in [2] all beams are divided in eight groups. This division leads to the entire volume being searched once every eight scans if all tasks are performed regularly.

Giving a fixed number of tracking tasks is impossible. The temporal demands however of the different track tasks can be defined. Of all tracking tasks TWS has lowest demands because this task translates into a VS task which is performed regularly. Only when a VS task is about to be dropped the TWS gives that particular VS higher priority.

The number of tasks for a TLP and a THP are based on a four second update rate and a one second update rate and their maximum number of tasks is denoted as NLT and NHT respectively. Confirmation tasks also have a maximum value denoted as NCT. Adding all these values leads to the maximum number of tracking tasks per scan, $NTs = NHT + NCT + NLT/4$.

The radar system itself has of course limited resources per scan. Since the algorithm gives higher priority to HS and tracking, these tasks are scheduled first. Only the remaining load is used for VS. The optimal amount of load for VS is available when no tracking tasks are needed. The VS tasks are divided into as many groups that fit in this optimal solution on a scan to scan basis, in this case eight.

When tracking is initiated the unperformed (low elevation) VS tasks are placed in a queue. At the next possible time, tasks from this queue are scheduled with priority over regular (low elevation) VS tasks, leading to an increasing queue length when the tracking load is constant. This is solved by allowing the queue to reach the length equivalent to a single scan. At that time the next entire scan is used to emit all queued tasks. After emptying the queue the scheme continues with queuing VS tasks to allow scheduling of track tasks. Implementation of this algorithm leads to the prioritisation: 1. HS, 2. tracking, 3. high elevation VS, 4. queued VS and finally low elevation VS.

3.2 Scheduling based on matrix representation of the sky

Duron and Proth, [7], also discuss task scheduling for a rotating MFR but base their scheduling algorithm on different principles. The scheduler of Duron and Proth is based on the environment whereas Barbato and Giustiniani base it directly on systems constraints. Another difference is the prioritisation of tasks.

Duron and Proth identify three task types in the RSP, probing (search), tracking and confirmation. Furthermore they divide the space under observation in a matrix of *n* rows and *m* columns. Because the radar rotates it can only see *k* columns at a time (*k*<<*m*). A temporal demand is placed on tasks. On probing this is done by ensuring that each cell is probed once every two periods. When a probing task detects an object, a confirmation task is requested.

Temporal demands are placed on the maximum time between detection and confirmation. With tacking the temporal demand is based on the needed update rate to follow an object. Priorities are therefore stated as: i) confirmation, ii) tracking and iii) probing.

Tracking tasks have more temporal demands and are therefore described in more detail. Each tracking tasks consists of twelve sub-tasks having constraints on when they should start. Due to all these temporal constraints, time is the key element in this algorithm for planning tasks. Calculating time windows for tasks is therefore very important. To calculate windows for confirmation and probing Eq. (3.3) and (3.4) are used. Each sub-tasks of tracking is considered to have the same window as a confirmation task.

In the scheduling algorithm the power constraint is ultimately used to see if a task will be scheduled or not. In order for this to work, knowledge is necessary about the power needs of tasks in time. Each task causes the power need to increase linearly by factor α, which is a parameter for accuracy. The total available power at any given time is constrained by a constant *W*. When idle and during reception periods the power demand will decrease linearly with factor 1. Requested tasks are evaluated based on power demands. If a task causes a power overload, it is postponed. When it can't be planned within its temporal window and the energy constraint, the task is dropped. Otherwise it is planned according to the energetic and temporal demands.

$$\begin{aligned} T_C &= R_C = kR^4\sigma \\ I_C &= \max\left(0, \frac{2R - c \cdot T_C}{c}\right) \end{aligned} \tag{3.3}$$

$$T_P = k(X+Y)^4\sigma$$
$$R_P = \frac{2Y}{c} \tag{3.4}$$
$$I_P = \max\left(0, \frac{2Y - a_S c}{c}\right)$$

In Eq. (3.3) and (3.4),

T_C : Emission period for a confirmation task;

R_C : Reception period for a confirmation task;

I_C : Idle time for a confirmation task;

T_P : Emission period for a probe task;

R_P : Reception period for a probe task;

I_P : Idle time for a probe task;

k : Constant representing power loss in the system;

R : Range of the target;

σ : Radar cross section of target, estimated by worst case scenario;

c : Speed of light;

X : Distance between radar and closet point in the cell that is observed;

Y : Distance between the closest and furthest point of the probed cell;

a_S : Weighted combination of T_P and R_P;

W : Power constraint within sensor.

The resulting algorithm updates the schedule each time a new column appears in the visibility domain. It starts by scheduling confirmation tasks that have to be done before that column gets out of visibility range. Then all tracking tasks are scheduled within the time the column is in view. Finally the probing tasks for the column are scheduled. When tasks cannot be scheduled the algorithm will try again when the next column appears since the column is still in view then. The second time these tasks are to be scheduled they have similar priorities, e.g. a previous unscheduled tracking task has lower priority than a new tracking task but higher priority than a new probing task. Any idle time that is still left while the column is in view is used for additional probing tasks.

Duron and Proth also give results of the algorithm for experiments with interleaving and non-interleaving strategies. They show that interleaving tasks leads to but some improvement in decreasing the radars idle time. Highest improvement is seen when the radar is heavily loaded. The number of objects that can be tracked is the main difference between the two strategies. This is logical because interleaving is developed for tracking.

The obtained results in [7] show that this algorithm supports over 300 tracks while still performing all search tasks. Much work however can still be done for improvement on the precision of the radar. Currently APAR is able to track little less than 300 targets but it had to drop all search tasks to do this. This shows that intelligent scheduling can improve radar performance tremendously, although this comparison is based on a simulation versus practical results and a rotating versus a fixed MFR.

3.3 Queuing of dwell requests

In [8] Huizing and Bloemen present an algorithm for dwell scheduling in a MFR. Input to their scheduler are the dwells to be emitted, so tasks are chosen and the scheduling is based on the dwell that are needed. This is different from the view in section 2.1 and 2.2 were they were separated. The underlying principles of this scheduler however could still be interesting for task scheduling.

Two types of dwells are identified in the process. First the normal dwell which is emitted for search, track, confirmation and midcourse guidance. The other contains dwells for TI. These are treated differently because of the importance of their time constraints i.e., TI has to be done at the right time. All dwell requests consist of dwell length, transmission window (earliest, desired and latest time of execution), ID number and priority (where this priority originates from is not discussed in [8]). Admission to any of the queues is only given when the current time is within the transmission window and the summed dwell length of the queue does not exceed the maximum queue length.

All requests from the CMS are placed in the request list according to the type of task. Each frame has its own branch in the request list. Branches are ordered by their priorities and requests within a branch are ordered by desired time of transmission. Based on this ordering the dwell request processor places dwell requests in the queues. When a request exists in the TI queue the queue length of the normal dwells is shortened. The new length is based on the available time between the interleaved TI dwells. When the TI queue is emptied the normal queue length is restored to its regular size. In figure 3.1 the basic outline is given for this dwell scheduler.

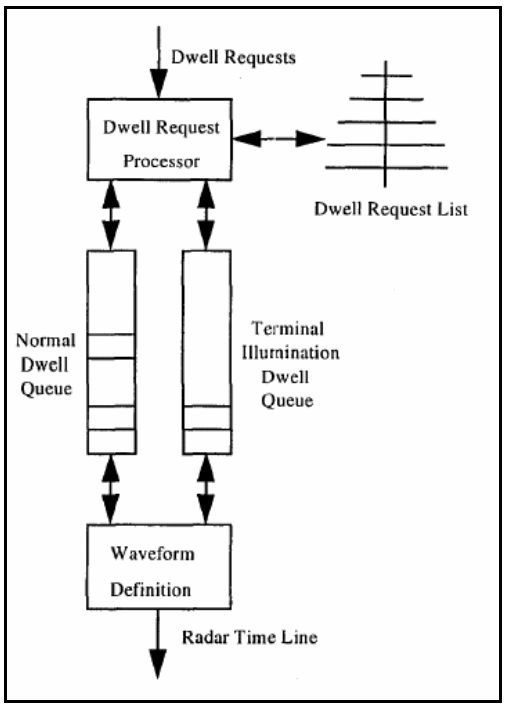

Figure 3.1 Outline of dwell scheduler proposed by Huizing and Bloemen, taken from [8]

3.4 Scheduling with intelligent agents

Thaens proposes a multi agent system (MAS) to solve the RSP for a MFR. In [21] the outline for this system is given. Each task is assigned to an agent that has the responsibility to reserve time on the radar time line for his task. Agents can place tasks on empty time slots or remove tasks with lower priority. An additional evaluating agent checks if the new schedule is better than the last one.

This evaluation is done by calculating the utility (sum of priorities of scheduled tasks divided by sum of priorities of all tasks) of the proposed schedule and compare it with that of the previous schedule. The one with highest utility is chosen as radar time line.

Thaens also discusses the necessary developments to optimise results with the MAS. The current development in optimising the results are done via negotiating strategies based on game theory [22]. In the negotiation agents try to find Nash equilibriums. This point is the local optimum for the agent's utility at a given time regardless of other agent's decisions. Tests on how well this scheduling methodology works in scheduling settings however are not yet available.

3.5 Cons of current techniques

Sections 3.1 – 3.4 introduced three developed schedulers for MFR and a new approach to a MFR scheduler, this section discusses their cons based on the problem described in section 1.1. The first scheduler is based on grouping the tasks based on function type and giving priorities to different types of functions. Although this leads to a fast scheduling algorithm it has some downsides. By giving HS the highest priority it is possible that high elevation tasks are dropped, which is not always preferable in certain missions and/or environments. Furthermore the sensor functions for missile guidance are neglected. Implementing these into the schedulers will probably require much research so looking for other methodologies that are more suited for this might prove useful.

The second scheduler is developed for a rotating MFR which is very different from scheduling a fixed MFR. The principle of this scheduler is also based on prioritising the types of functions rather than the specific tasks and this scheduler also neglects missile guidance tasks. The calculation of the task duration also has some negative sides to it. Firstly, it uses the radar cross section of the object. In real applications this value is extremely hard to estimate because the classification of an object could be unknown and because the radar cross section is highly dependent on the angle with which the radar wave is reflected by the object. Secondly, by using defined probe areas information could be lost. When the time is available the reception window should always include the entire range of the radar. This ensures that objects are detected even when they are just outside the probe window.

The other two discussed schedulers are based on task specific priorities. The first of these is the dwell scheduler. Downside to this scheduler is that it can not be expanded to scheduling sensor networks. This causes difficulties in fixed PA MFRs like APAR. This radar consists of four radar faces that should be scheduled as four separate sensors, each covering its own volume of space. What happens with this scheduler is that the sensor is seen as a whole so when one face is executing TI tasks, the other faces are unable to execute search tasks. Furthermore, it uses interleaving for TI tasks, which can not be performed by many sensors yet.

The answer to all the problems mentioned above could be to use MAS for sensor scheduling. This methodology can deal with set-up times and sensor networks. Downside is that much more research is needed on the underlying structure and negotiating schemes before tests can be done with this scheduler. Using already developed schedulers from other domains might be a way to find a good scheduling algorithm for solving the RSP with less research effort.

3.6 Summary

Solving the RSP for rotating MFR is considerably different from solving it for fixed MFR. Most of the developed intelligent radar schedulers are however based on rotating MFR. The use of object specific priorities and set-up times in the scheduling system are neglected in most practical systems. An observation on current radar schedulers is that they are not yet applicable for sensor networks. A scheduler based on MAS could prove to work well for scheduling sensor networks but needs more research. Looking at existing schedulers and using their general approach on the RSP for a sensor network might lead to a good scheduling algorithm that can be implemented in practical systems with less research effort since adapting existing methodologies in resembling domains probably is less work than designing an entirely new algorithm.

4 Overview of applicable scheduling techniques

Sections 4.1 – 4.7 will introduce and discuss several existing scheduling algorithms for general problems. In the final section, 4.8, interesting techniques for solving the RSP are identified.

4.1 Fuzzy Lyapunov synthesis for scheduling

Margialot and Langholz [12] propose a way of designing schedulers based on fuzzy Lyapunov synthesis (computing with words). The result of this approach is tested on a single machine job shop scheduling problem (JSSP). Within the job-shop setting different types of parts exist and the machine requires a set-up time between processing parts of different types. Buffers are formed of parts waiting to be processed. The resulting scheduling algorithm is based on the size of these buffers. However, the number of parts in a buffer is not the only factor in the scheduling algorithm. To distinguish between different parts the processing times are taken into account as well.

Known policies such as Clearing Policy (CP), Clear Largest Buffer (CLB) and Clear-a-Fraction (CAF) are used as comparison to the fuzzy schedulers result on a benchmark which is discussed in [12]. The policies all calculate from which buffer the next parts should come. In [12] and [16] these policies are explained in detail. In this thesis only the performance measure is given. Performance is evaluated on the average load of weighted buffer size ($\overline{B}_w$) and is given by Eq. (4.1).

$$\overline{B}_w = \frac{1}{t}\int_0^t \sum_{\forall i} \tau_i \cdot x_i(s) \cdot ds \tag{4.1}$$

In Eq. (4.1), x_i is the buffer size of type i, which is a function of time, and τ_i is the processing time needed for one part of type i. The goal of schedulers is to minimise this function. Margialot and Langholz achieve this by filling a rule base with the results of a linguistic function. Derivation of this function is based on stability analysis of the scheduled system.

According to Lyapunov a system is stable if the derivative of the system function is negative. Information on Lyapunov synthesis can be found in [4] and [24]. The system function V and the resulting linguistic function are given in Eq. (4.2).

Only one $\dot{z}$ within the entire set of i can be negative, i.e. only one type can be processed at the same time leading to a decrease of buffer size. Since the goal is for Eq. (4.2) to be negative, the rule base will enforce that part of type i will be processed with large z_i. Rules are made based on the weighted buffer sizes, e.g. if buffer 1 is big and buffer 2 and 3 are small, process parts from buffer 1. Membership functions are used to determine what a small or large weighted buffer size is.

$$\begin{aligned} V &= \frac{1}{2}\sum_{\forall i} {z_i}^2 \\ \dot{V} &\cong \sum_{\forall i} z_i \cdot \dot{z}_i \end{aligned} \quad \text{, with} \quad z_i(t) = \tau_i x_i(t) \tag{4.2}$$

A good comparison can be made between the JSSP and the RSP, especially when the one machine case is compared to the single sensor problem. The generalisation from the one machine set-up to a multi-machines set-up can also be used when generalising the RSP from one MFR to a sensor network.

Using fuzzy schedulers based on Lyapunov synthesis could prove useful in solving the RSP because buffers are bounded and minimised while no jobs are rejected. The theory even allows for set-up times between different types of jobs which is also the case in a MFR when switching between a TI task and another task. Another plus of this approach is the low level of a priori knowledge needed to construct the scheduler.

One problem however needs to be solved before this scheduling technique can be used for solving the RSP. The weighting of buffers needs to be altered to fit the RSP. Furthermore, this scheduler works on machine demand, meaning that it is rather greedy: it does not optimise the entire schedule; it chooses a task at a given time. This could lead to sub-optimal performance in overload situations.

4.2 Scheduling with gatekeepers

In [11] Lin describes a system for admission control based on a game theoretic model. The environment consists of a server, gatekeepers and incoming requests directed to the server. The server is shared by different applications so the incoming requests are from different directions, each direction having its own gatekeeper. The gatekeepers determine whether a request is allowed to the server or not. While performing his task the gatekeeper does not know whether the server is busy or not and what the other gatekeepers are doing.

The problem a gatekeeper faces is that he doesn't know whether an allowed request can be handled by the server or not. On the other hand, he could reject a request while the server was able to handle it. The proposed algorithm is based on game theory i.e., the gatekeeper needs to get as many points as possible. Points are given for each request that is served and points are deducted for all admitted but unserved requests. In [11] several ways of optimising this problem are given.

When the different tasks of a MFR are viewed as applications using the same server, being the MFR, it would be feasible that using gatekeepers is an option in solving the RSP. The system proposed by Lin however does need some additional work to implement it in the MFR domain. In the current domain no points are deducted for rejected requests which would lead to performance decrease when used for the RSP. In order to get a high performance MFR the rejected and unserved request must be minimised. Before application the optimisation formula should be expanded to include penalties for rejection, otherwise a tracking task for an incoming missile could be dropped because the system is tracking an airliner.

4.3 Neural network scheduling

In a lot of JSSPs a queue is made of jobs that are waiting to be processed. Optimising the performance of a machine can be done by ordering the queue based on job contribution to overall performance. In [1] Alifantis and Robinson present a methodology to do this with a NN. Visual simulations of small size JSSPs are done in which the user/expert has to make the scheduling decisions. The resulting schedule and the input are stored in a database.

The complete dataset is then used to train a NN. Since the trained NN can generalise, optimal solutions should be found to similar but larger sized JSSPs. Actual schedulers based on this principle have not yet been implemented and good comparisons with other scheduling methodologies is therefore not possible.

Applying the NN based methodology in scheduling to the RSP gives three problems. Firstly, scheduling in a MFR is usually done at run-time and scheduling with a NN calls for a matrix (vector) representation of tasks and schedules. Using time intervals for which schedules are made in advance will overcome this problem. The size of such a time interval should be examined through experiments. Second problem is obtaining the training data. A simulation model would have to be created in which the user makes the schedules. The resulting NN will depend on the user that worked with the simulation.

This problem is larger in the RSP than in the JSSP because in the RSP concepts of risk posed by detected targets are subjective whereas costs and processing time on machines are objective. Finally, the transparency of the resulting scheduler is minimal. This is mostly considered as an extreme negative in military applications. A possibility could be to use the resulting values of a trained NN as threshold values in a rule base. This would lead to an adaptive and fast scheduler with predictability.

4.4 Scheduling with genetic algorithms

Time intervals also have to be used when GAs are applied to optimisation problems. One of the major concerns when using a GA is to translate the possible solutions to chromosomes. Thilakawardana and Tafazolli [23] made a system based on a GA for data traffic.

The chromosome they propose has the length g (the number of resources) and are filled with tasks from n different service classes, with $n>>g$. Each service class has its own characteristics such as Quality of Service (QoS), dynamic queue length (q) and the resource frequency (f). The QoS for a class denotes the priority it holds in getting served by the system. Factors like arrival rate, duration and service rate are combined in q of a class. When these first two factors are the only factors taken into account for the fitness of a chromosome, a single class could use up all resources. That is the reason for f, the frequency of resources a class requests. Combining these factors leads to the fitness of a chromosome (C_F) as given by Eq. (4.3). Using this fitness function, the normal procedure of GAs can be used to finding an optimal schedule.

$$C_F = K \cdot \sum_{i=1}^{g} \frac{QoS_i \cdot q_i}{\sqrt{f_i}} \text{ with } K \text{ is a constant.} \tag{4.3}$$

Translating the chromosome mapping to the RSP domain implies replacing the number of resources by the amount of available sensor time. How to replace the different service classes however is not as easy. Of course QoS can be replaced by the priorities given to different tasks but these aren't constant within a single family of tasks.

Making groups out of tasks based on their priority would solve this first problem but another problem remains. This concerns the statistical information (resource frequency) about the requests. In a MFR this information is usually unknown and can vary enormously due to changing missions. To overcome this, the fitness function needs to be changed to fit the RSP domain.

A drawback of GAs still remains; this however has nothing to do with implementation issues. Finding optimal solutions with GAs is computational heavy and is therefore usually done off-line, this in contrast to the desire for an online, real-time and adaptive scheduler for solving the RSP. A possibility could be to have a GA optimising strategies in the background and use that information to update the scheduling strategies used online.

4.5 Constraint satisfaction programming in scheduling

In the literature several systems can be found that make schedules based on constraint satisfaction. In [20], [17] and [18] different approaches are discussed and evaluated. The schedulers proposed by Schild [20] work in the domain of communication between processors. When looking at this domain, assuming there is only one line of communication shared by all processors; a comparison can be made with the single machine JSSP. Important in this work is that there are two types of tasks, periodic and a-periodic. Schedules are then made for cycle times (*ct*). The length of *ct* is chosen such that all periodic tasks will stay in sync throughout all repletion windows. A-periodic tasks can now be assumed to be periodic with a period equal to *ct*. This approach is used because execution starts on the beginning of *ct* according to the schedule for that window.

The constraints are based on temporal effects like starting and finishing jobs within *ct* but are also based on domain characteristics. An answer from processor B cannot be scheduled before the sending of the question has finished. A number of search techniques are used for solving the set of constraints to create a schedule.

In [17] and [18] different approaches are introduced to deal with constraints. Schild [18] makes a schedule based on constraints; this scheduler finds a solution and then checks for constraint violations. Pinto and Grossman [17] propose an algorithm that is based on techniques from both operations research and artificial intelligence. The problem is divided in sub problems which are solved. Results are then evaluated based on the constraints. If the solution doesn't comply with the constraints the sub-problem is re-divided into sub-problems, this system keeps iterating until all (sub) problems are solved.

Policella et. al. [18] make a long-term schedule and then adjust it to deal with changes in the environment. They suggest two ways of accomplishing this. The first is to calculate a scheduling envelope stating the maximum number of combinations of different jobs. Each new job is evaluated based on this envelope before it is inserted. The second way is to look at the conflicting constraints that occur when jobs are scheduled based on their earliest start times and solve this.

Although constraint satisfaction produces good schedules, neither of these approaches is suited for solving the RSP based on their computational complexity. In solving the RSP the optimality of the schedule isn't the only criteria, the speed in which it is obtained is of equal (if not greater) importance. Using the schedulability envelope principle however is an option since all computations are done prior to run-time.

4.6 Sequence scheduling

In many applications the success of a schedule depends on the temporal and functional correctness. In [9] Hwang and Cheng propose an algorithm that works on sequences of tasks and then optimise a solution by minimising the number of rejected tasks and flow time. To obtain this goal two algorithms are created, the 'sequence scheduler' and the 'set sequencer'. Understanding these algorithms requires some basic knowledge of sequences and the concept of master sequences.

In both algorithms each task τ consists of three variables, the ready time r, the computation time c and the dead line d. All tasks to be scheduled are combined in task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. A task set is considered to be feasible when a schedule exists in which all tasks meet their timing constraints. A sequence is defined as $S = \langle \tau_1^S, \tau_2^S, ..., \tau_k^S \rangle$ with $k \leq n$. A sequence specifies the order in which the tasks are executed. There are generally two ways of constructing a sequence, (i) based on priorities or (ii) based on time constraints. In the two algorithms discussed here the criteria is time, which is the reason that the priority of a task is not included in the task definition.

The first algorithm (sequence-scheduler) finds an optimal sequence σ for any given sequence μ. Two assumptions have to be made in order to find an optimal schedule. The first assumption is that $r_i + c_i \leq d_i$ with $1 \leq i \leq n$ holds, which basically means that all individual tasks have to be schedulable.

The other is that σ need not contain all tasks in μ. Meaning that σ is optimal with degree j where j is the number of tasks in the optimal sequence. The algorithm evaluates sequences S with increasing values for j that conform[1] to μ and are feasible. The one with the shortest finish time is then chosen as optimal sequence and outputted.

The set-scheduler is based on the concept of super sequences and uses the sequence-scheduler to find optimal solutions. A super sequence Δ is defined as a sequence in which tasks from Γ can appear multiple times in such a way that every feasible scheduling sequence S conforms to Δ. Before this super sequence can be formed all tasks in Γ have to be divided in two groups, top-tasks and non-top-tasks. A top-task is defined as a task that doesn't contain any other task. Containment is defined as $\tau_i \subset \tau_j \quad iff \quad r_i < r_j \quad and \quad d_i > d_j$.

All top-tasks (a top-task is denoted h) can appear only once in the super sequence. The non-top-tasks containing top-task h_k are denoted as set M_k, all other non-top-tasks are denoted as set $\overline{M_k}$. The basic structure of a super sequence around the k-th top-task is given in Eq. (4.4). In this equation the operator $\oplus$ is used for concatenation of sequences.

$$\Delta = \left\langle B_{k-1,\overline{k}} \cup B_{k-1,k} \cup B_{\overline{k-1},k} \right\rangle \oplus h_k \oplus \left\langle B_{k,\overline{k+1}} \cup B_{k,k+1} \cup B_{\overline{k},k+1} \right\rangle, \quad where$$
$$B_{k,\overline{k+1}} = M_k \cap \overline{M_{k+1}} \quad , \quad B_{k,k+1} = M_k \cap M_{k+1} \quad , \quad B_{k,k+1} = \overline{M_k} \cap M_{k+1}. \tag{4.4}$$

Each subsequence B consists of non-top-tasks that have to be ordered. For the first type this is done based on deadlines, the last subsequence is ordered based on release dates and the middle is ordered randomly. This principle is illustrated by an example consisting of 10 tasks with 3 top-tasks, as given in figure 4.1 and the super sequence for this example is given in figure 4.2.

[1] Let τ_i and τ_j be two tasks belonging to a sequence S. If τ_i is located before τ_j in the sequence S, we say that $<\tau_i, \tau_j>$ conforms to S.

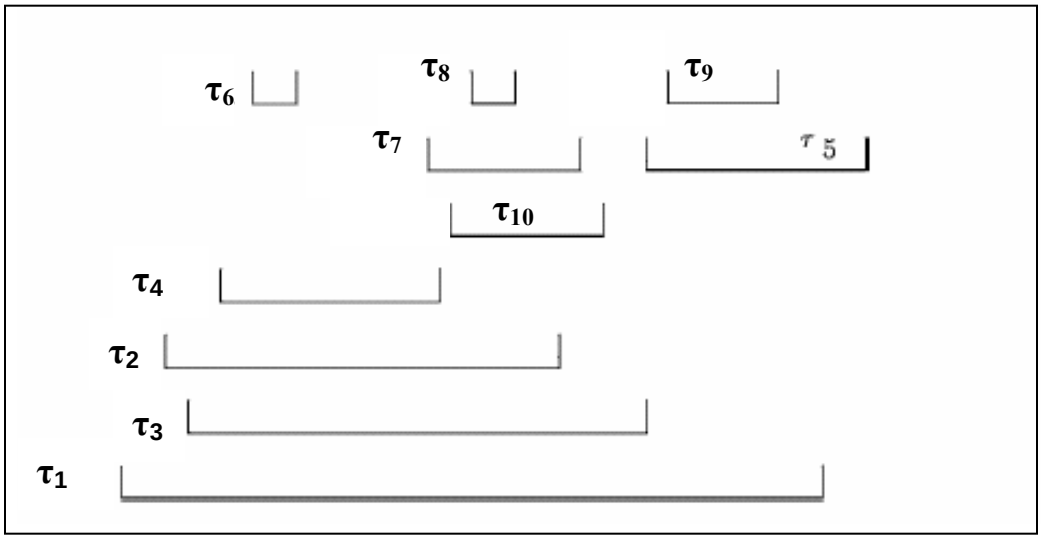

Figure 4.1 Ten tasks that need to be scheduled based on their ready and due dates

Sequences can be made with the same length as Γ that conforms to Δ. Each of these sequences is inputted to the sequence-scheduler and their results are compared. The output of the set-scheduler will be the schedule with the most tasks. If two or more schedules contain the same amount of tasks the one with the shortest processing time is chosen as optimal solution.

$$\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9, \tau_{10}\}$$

$$h_1 = \tau_6 \quad , \quad h_2 = \tau_8 \quad , \quad h_3 = \tau_9$$

$$M_1 = \{\tau_1, \tau_2, \tau_3, \tau_4\} \quad , \quad M_2 = \{\tau_1, \tau_2, \tau_3, \tau_7, \tau_{10}\} \quad , \quad M_3 = \{\tau_1, \tau_5\}$$

$$\overline{M_1} = \{\tau_5, \tau_7, \tau_{10}\} \quad , \quad \overline{M_2} = \{\tau_4, \tau_5\} \quad , \quad \overline{M_3} = \{\tau_2, \tau_3, \tau_4, \tau_7, \tau_{10}\}$$

$$\Delta = < \tau_1, \tau_2, \tau_3, \tau_4, \underbrace{\tau_6}_{h_1}, \underbrace{\tau_4}_{B_{1,2}^-}, \underbrace{\tau_2, \tau_3, \tau_1}_{B_{1,2}}, \underbrace{\tau_7, \tau_{10}}_{B_{1,2}^-}, \underbrace{\tau_8}_{h_2}, \underbrace{\tau_2, \tau_7, \tau_{10}, \tau_3}_{B_{2,3}^-}, \underbrace{\tau_1}_{B_{2,3}}, \underbrace{\tau_5}_{B_{2,3}^-}, \underbrace{\tau_9}_{h_3}, \tau_1, \tau_5 >$$

Figure 4.2 Construction of super sequence for problem illustrated in figure 4.1

Although this scheduler finds optimal schedules it has several characteristics that could make it less suited for solving the RSP. Firstly, the set-scheduler is very computational heavy. It uses the sequence-scheduler, which has complexity $O(n^2)$, where n is the number of tasks, for N sub-sequences of Δ resulting in complexity $O(N \cdot n^2)$. In this measure for complexity the value N depends on the number of tasks and the amount of top-tasks within the task set. A solution could be to choose a small n when applied to the RSP. This however will lead to many computations each time interval.

Another problem is the set-up time that has to be considered for the radar when switching between different types of tasks e.g., search and illumination tasks. Including these set-up times to this scheduling technique would lead to even more computation demands.

In [6] Dauzère-Pérès and Sevaux solve a single machine JSSP with a master sequence. The master sequence M is constructed by the algorithm in figure 4.3, the task set is denoted as J and a task on position i in J as J_i. It can immediately be seen that the master sequence is larger than the super sequence.

To reduce the complexity of trying all schedules conforming to the master sequence they propose a Lagrangean relaxation to eliminate tasks in the master sequence. The optimisation is solved with a mixed integer programming method. Doing so, [6] shows that the problem complexity is $O(P)$, with P is the length of the (reduced) master sequence. This type of solution could prove useful in solving the RSP as long as the number of removed jobs doesn't pose a threat to the performance of the MFR.

```
FOR every job Ji ∈ J    DO
     M ← M∪Ji
     J ← J∪Ji
     FOR every job Jj ∈ J and Ji≠Jj  DO
           IF dj ≥ di DO
                 M ← M∪Jj
           ENDIF
     ENDFOR
ENDFOR
```

Figure 4.3 Algorithm to construct master sequences

4.7 Multiple families of jobs in the JSSP

As mentioned several times before, most schedulers assume a zero set-up time between different kinds of jobs. In many applications this however isn't the case. To create a scheduler some definitions of concepts have to be given in order to model the algorithm. Definitions here will follow the notation used by Chen and Powell in [5].

A set of jobs, denoted as *J*, consist of *n* different jobs that have to be processed on *m* parallel machines. There are *b* families and each family *u* has a set J_u, consisting of n_u jobs with $J \subseteq J_u \quad 1 \le u \le b$, $n = \sum_{u=1}^{b} n_u$ and $J = \bigcup_{u=1}^{b} J_u$. The set up time s_{uv} is defined as the time needed when a job of family *u* is followed by a job from family *v*. The initial set up time is considered to be zero ($s_{0u} = 0$). When considering set up times in scheduling two different types of set up times need to be considered.

First type is the sequence independent set-up time, i.e. $s_{uv} \equiv s_v \quad \forall u$, $1 \le v \le b$, $0 \le u \le b$, and $u \ne v$. Second type of set-up times is sequence dependent, i.e. set up times depend on the first and the second family. Normally the inequality $s_{uv} + s_{vw} \ge s_{uw}$ $\forall \quad u, v, w$ is assumed in the sequence dependent case. Optimisation criteria are based on the same principles for each case, one criterion is minimizing the weighted completion time and the other is minimizing the weighted number of tardy jobs. In [5] Chen and Powell propose a column generation based B&B method leading to an exact solution for medium sized multiple job scheduling problems on parallel machines. (Medium sized in this sense is $n = 40$, $b = 6$ and $m = 4$).

Results obtained by this algorithm and the specific details of implementation can be found in [5]. Solving the medium sized problem on a Sun workstation with a 164 MHz processor was done in average well under 10 seconds. Optimizing weighted completion time was solved in an average of 8 seconds, optimizing the weighted number of tardy jobs was done under 2 seconds on average.

In general, the algorithm works by formulating the problem as a set partitioning type formulation with an exponential number of columns. This is solved by the branch and bound approach. In each iteration of this branch and bound the columns are generated by decomposing a linear relaxation into a master problem and sub-problems which are solved.

4.8 Choosing a proper scheduling technique

This section gives an overview of the discussed algorithms. The focus lies on the usability in solving the RSP. All systems are therefore considered on their computational load, ability to translate domains, utility of resulting schedule and transparency. The latter is of importance because operators need to understand the system in order to evaluate performance. Another reason is the ability to predict the reaction of the system in extreme circumstances. Results of the different systems based on those criteria are given in table 4.1.

Based solely on speed the choices would be to use Lyapunov synthesis, NN scheduling or admission gatekeepers. Speed however isn't the only criterion. Rejection of tasks means a decrease in utility. This can be dangerous in the military domain if the rejected task concerned an incoming missile. Admission gatekeepers have a high rejection rate and are therefore not suited for solving the RSP.

Predictability of the system is highly valued in the military. Operators want to know how the system will react in different situations. The use of NNs doesn't comply with that wish. Another problem of NNs is getting the appropriate training data and choosing appropriate in- and output variables.

This leaves only the scheduler based on fuzzy Lyapunov synthesis. Looking at the pros and cons of this system this one looks most promising. It has low computational needs when used online and set-up times can be introduced. The only problems that need solving are: i) adjust the formula for buffer weights and ii) find membership functions to construct the fuzzy rulebase.

Table 4.1 Pros and cons of different P&S algorithm for solving the RSP

System	Speed	Domain Translation	Transparency	Utility	On- or off-line
Fuzzy Lyapunov	+	+	+	+/-	ON
Admission gatekeepers	+	+	+/-	-	ON
NN scheduling	+	+	-	+/-	ON
Genetic Algorithms	-	+/-	+/-	+	OFF
CSP with an envelope	-	+/-	-	+	OFF
LCP	-	+	+	+	ON
Sequence scheduling	-	+	+	+	OFF
JSSP with job families	-	+	+/-	+	OFF

Other possibilities can be found in the combination of different approaches. It is known that GAs are very good in finding solutions for problems with a large search space. The downside however is that they're not that fast in finding the solution. This problem can be overcome by using a GA in the background. In such a configuration the results of the GA are used to update the scheduling strategy used on-line. Running a GA in the background ensures that the strategy adapts to a changing environment. This ability to adapt is considered valuable during military missions.

A GA could be used to generate training data for a NN. Having the GA retrain the NN based on recent task sets makes the overall system adaptive to the environment. The only problem that remains then is the non-transparency of the overall system since both NNs as well as GAs have low transparency. Furthermore, this approach requires the tasks to be divided into vectors in order to use the NN. This however could be a large problem if tasks are requested on a regular basis.

Another option could be to use a sequence scheduler as off-line optimisation tool in stead of a GA. The choice for the GA is supported by the problem size. Scheduling small task sets leads to lower utility than scheduling large[2] task sets. Sequence scheduling is based on comparing all possible schedules which takes a long time when the search space grows. Speed in the off-line scheduler is considered since it determines the update rate of the NN and so directly influences the adaptability of the online scheduler.

A third option is to use a GA online. The GA bases the fitness of a schedule on a buffer with fixed size. The number of generations is set to infinite and the best schedule adapts to current buffer entries. Execution of tasks is only based on the first entry in the current best schedule when a sensor is available to execute it. In case the new schedule isn't calculated in time when a sensor requests a task, the second entry of the previous schedule can be considered to be sub-optimal.

Combining scheduling techniques can be examined further. When using the fuzzy Lyapunov based scheduler, the NN scheduler and the online GA, hybridisation is possible that takes the best solution of three possible solutions. In this particular case the NN solution is trained based on the GA and can therefore be excluded in the hybridisation process. Other heuristics used in scheduling can be added to the hybrid scheduler in order to obtain some guaranteed performance.

Considering these possibilities leads to the conclusion that four possible schedulers are to be implemented and tested

i) Fuzzy Lyapunov synthesis;
ii) Using a GA off-line to train a NN which is used online;
iii) Using the GA in an online setting;
iv) A hybrid scheduler which combines scheduling heuristics with the fuzzy Lyapunov and the online GA.

[2] A set is considered large if it consists of more than 10 tasks.

For each of these schedulers a hypothesis can be made based on the description given in this chapter.

(1) Scheduling based on Lyapunov synthesis results in a good trade off between performance, processing time and system predictability.
(2) Scheduling with a neural network (NN) trained by a genetic algorithm (GA) will result in an adaptive NN that has low processing time, medium utility and low system predictability.
(3) Scheduling using a GA online will comply with the real time demands of naval warfare but has a low predictability and no guarantees can be made on performance, although it is expected to have a high performance.
(4) Hybridisation of different scheduling techniques will lead to a fast algorithm with a guaranteed minimum performance.

5 Modelling the schedulers

Based on the results of chapters 3 and 4 this chapter will introduce three models for solving the RSP. In section 5.1 the radar scheduling process is placed in the command and control loop. The detailed problem description of the RSP and the required data flow are given in 5.2. Sections 5.3 – 5.5 describe the fuzzy Lyapunov based scheduler, the GNN scheduler and the online GA scheduler respectively. An introduction to make a hybrid scheduler based on the scheduling models is given in section 5.6. The measures on which evaluation can take place are given in section 5.7. Theoretical background on the used techniques is given in appendix A.

5.1 Overview of the entire system

The radar scheduler is an integrated part of the entire command and control loop. Before models can be made of scheduling principles this loop must be described. The command and control loop is illustrated in figure 5.1.

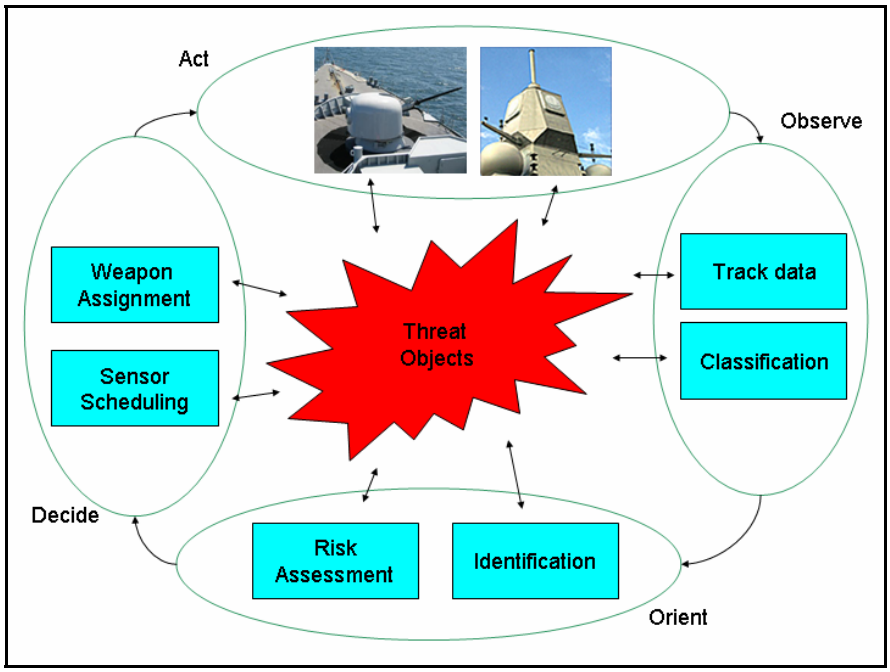

Figure 5.1 Command and control loop

The control process is based on the Observe-Orient-Decide-Act (OODA) loop. It starts with observing the environment with sensor measurements. Based on tracking data obtained by sensors, classification of the detected objects can be done. After observing the environment orientation has to be done for threat assessment. This means that objects need to be identified and their threat to the mission evaluated.

Mertens [13] deals with the classification and identification problem. She proposes Bayesian belief networks to classify and/or identify objects based on sensor data, intelligence reports and rules of engagement.

Risk assessment, as proposed by Bolderheij and Van Genderen [3], uses, amongst others, the results of the classification and identification process. Based on the risk assessment priorities can be given to tasks belonging to detected objects. A good feature of the risk assessment process in [3] is that search tasks are represented as imaginary objects based on intelligence reports of threats in the environment. This means that priorities can be given to search tasks based on the same principles as track tasks.

Based on the orientation process, decisions are made regarding sensor tasks and weapon deployment (both hard- and soft-kill). Of course the latter influences the first due to weapon guidance and target illumination. Operator influence exists most in the decision making process for weapon deployment. The decision making for radar is too complex for direct operator influence and is therefore subject to automation. The decisions made lead to radar emissions and possibly weapons fire.

Using this OODA loop has the advantage that it can be generalised to a multi sensor platform. The sensor scheduling is first done on network level (which tasks are performed by which sensor) and later on the sensor level (which task is executed when).

Note that the processes in figure 5.1, although modelled based on the OODA-loop, are no longer a loop in this new approach. All processes are executed parallel based on the available information of the objects, both imaginary and real. It can therefore be seen as MAS with a blackboard. All processes are agents and the information on objects is placed on the blackboard.

In section 2.1 a general description of the RSP is given. The problem described there is valid for a single MFR assuming that the CMS makes the task requests. With the object oriented and parallel approach to the sensor control loop given a new view on sensor[1] scheduling can also be made in more detail. In the new parallel approach, task requests are made based on the threat store of imaginary and detected objects. Since all objects are now combined in a single store means that each task is initiated based on an object. Following through on this concept, resource allocation needs to be done in order to assign tasks to the most suited sensor. These decisions are made based on task type, position of the object, detection probabilities and availability of the sensor.

The concept of this resource allocation can easily be generalised from a single ship configuration to a network of different sensors in NCW, assuming that the required communication needs are available. Scheduling of tasks on a single sensor level can be seen as a different problem, stated in section 2.1. This section also stated that dwell scheduling is considered to be manufacturer depended which is also assumed here. Figure 5.2 shows the resulting sensor loop based on a threat store. Since this study focuses on a single MFR configuration the complexity of resources allocation is neglected.

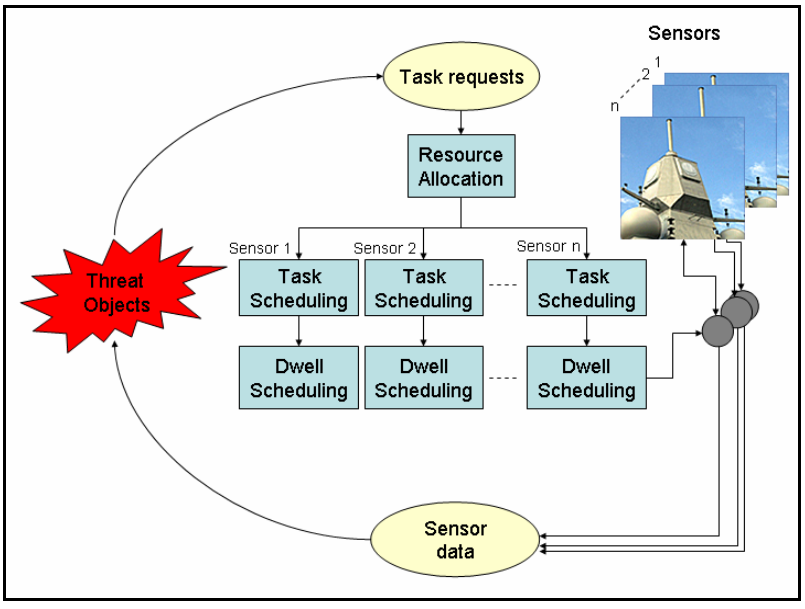

Figure 5.1 NCW sensor loop

[1] In a network setting the term sensor rather than radar is used to emphasise that all sensors are included in the scheduling process.

In this sensor loop the influence of sensor availability is neglected to identify the difference in the process of resource allocation and task scheduling. To reach optimal schedules these processes need to be integrated.

5.2 Data flow in sensor scheduling

The relevant processes in sensor scheduling were given in section 5.1; these however don't give enough information to implement schedulers. The flow of data, rather than the order of processes, is needed for the development of algorithms. The general view on the environment of sensor scheduling is given in figure 5.3.

Figure 5.3 shows that the sensor scheduling functionality communicates with three entities. The operator can influence scheduling based on mission objectives and emission directives. The sensors are observed to determine their availability to receive new tasks. The third 'user' of the scheduler is the weapons system. Depending on the status of own missiles in flight the sensors need to execute emissions to direct them.

In section 5.1 the object-oriented approach of Bolderheij was introduced, this concept in the sensor scheduling environment leads to a more detailed environment of sensor scheduling as shown in figure 5.4. All proposed algorithms are based on this data flow decomposition of the sensor scheduling problem.

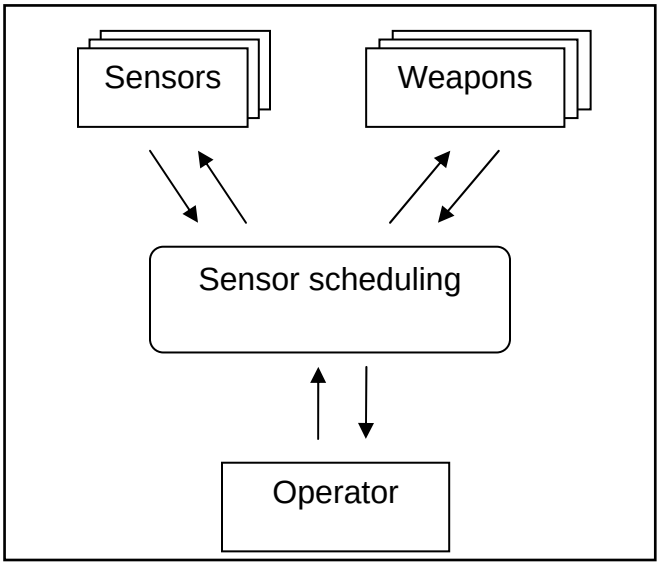

Figure 5.3 The context diagram for sensor scheduling

The importance of the object store can be seen in figure 5.4. All task requests are based on the information in that store. The goal of the 'request' functionality is to minimise uncertainty in object attributes (such as classification and identification) and to maximise the mission statements. Status of the systems is naturally included as input to that function to ensure valid requests and maximising the performance of weapon deployment.

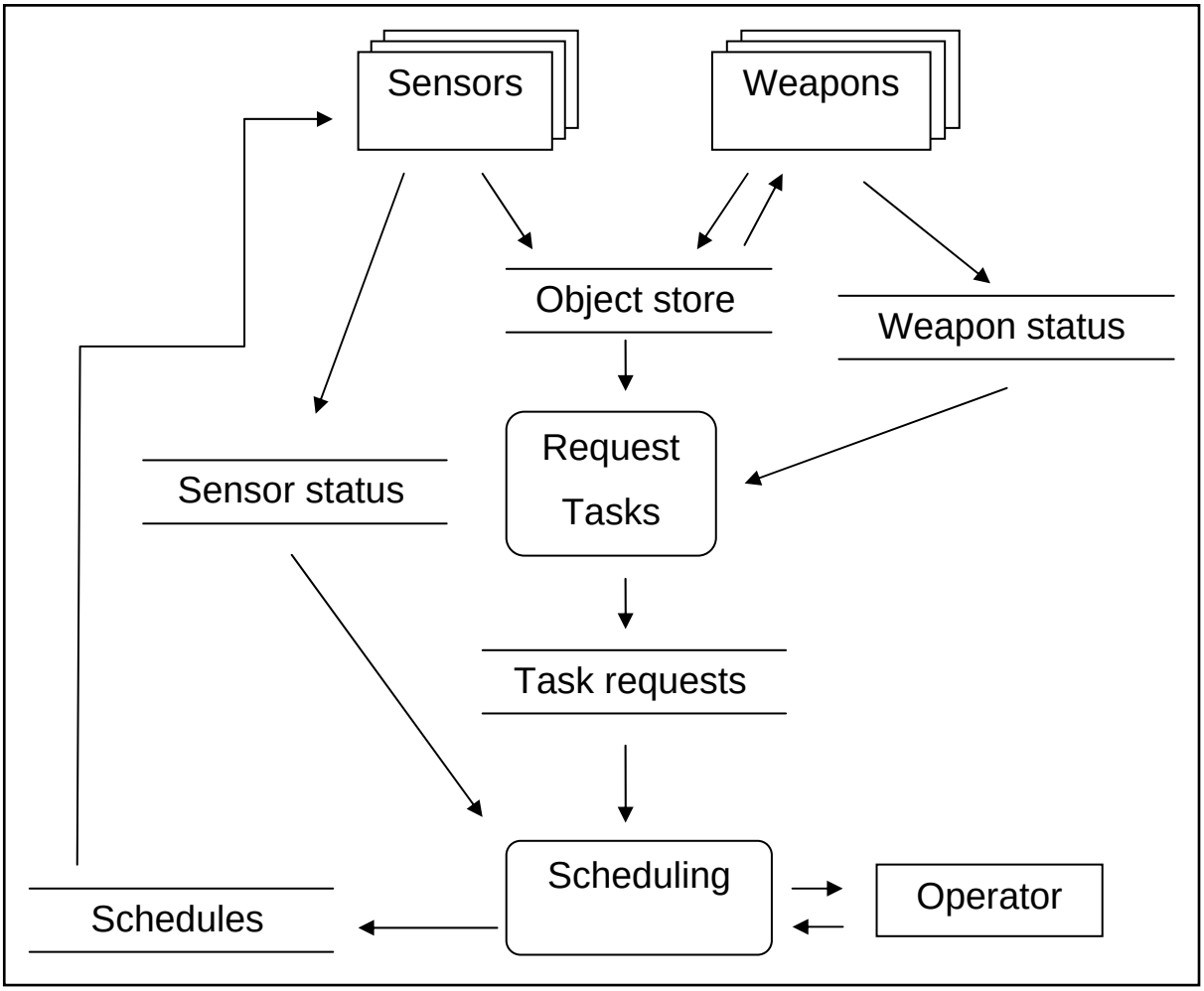

Figure 5.4 Data flow for task generation in sensor scheduling

In discussing the different schedulers it is assumed that all tasks are generated based on the principle of figure 5.4. It is therefore assumed a task request consists of at least:

- Type of sensor function;
- Pointer to or data on the object it refers to;
- Accuracy demands;
- Temporal demands, such as due date and processing time;
- Priority based on risk assessment.

This thesis does not focus on the entire command and control process so referring to the object is not necessary. Furthermore, the focus lies on the concepts of scheduling methodologies. Tasks can therefore simply be represented by their sensor function type, priority, due date and the task duration.

Most schedulers currently in use include a ready-date in their task definition. Here this feature is not used since tasks are requested based on the object store and can therefore be executed at first on the same time it was requested.

5.3 Fuzzy Lyapunov based scheduler

Based on figure 5.2 task scheduling for a single sensor can be seen as placing tasks sequentially on a single timeline. This means that execution of all tasks will never exceed the physical constraints of the radar. Tasks however are not always requested in their execution order. All requests are therefore placed in buffers. A fuzzy scheduler is used to determine from which buffer tasks are to be processed. According to the RSP given in section 2.1, this system will work with four buffers one for each task type. For each buffer the weight of this buffer (also called buffer size), B_W, is calculated with Eq. (5.1).

$$B_W = k \cdot \sqrt{X} \cdot \sum\nolimits_{\forall i} \rho_i P_i \left(\frac{(1-\alpha)\cdot \tau_i + \alpha}{\gamma \cdot \left(t_{dd,i} - t_{ct} - t_{td,i}\right)} \right) \tag{5.1}$$

The following denotations are used in Eq. (5.1):

k	:	Weight factor of the buffer;
X	:	Number of tasks in the buffer;
ρ_i	:	Range factor of object i, 1 if in range of the sensor else 0;
P_i	:	Priority of task i in the buffer, with $0 \le P_i \le 1$;
τ_i	:	Relative processing time in the buffer of task i, with $0 \le \tau_i \le 1$;
$t_{dd,i}$	:	Due date of task i in the buffer;
t_{ct}	:	Current time;
$t_{td,i}$	:	Processing time of task i in the buffer;
α	:	Weight factor to map relative processing time to a different domain, $[0,1] \to [\alpha,1]$, with $0 \le \alpha \le 1$;
γ	:	Weight factor for due date, with $0 \le \gamma \le 1$.

The weight factor per buffer is used to keep track of how well a sensor is suited for that particular function type. All sensors therefore have their own set of k-values, each set containing the values for the different function types. Doing so means that even if a sensor is not best suited for a particular function; it can still execute such a task if the primary sensor for that function is saturated. Besides these different k-values, the different sensors also have other values for τ_i and $t_{td,i}$, it would therefore be best if the weights of the different buffers are maintained separately for each available sensors.

Additionally, this weight factor can ensure that certain function types are given additional priority over other function types. If e.g. there are few weapon guidance tasks (X is small), the Bw value can still exceed that of filled buffers of search or track tasks (large X) by means of the difference in k-value. The general scheduler will operate as illustrated in figure 5.5.

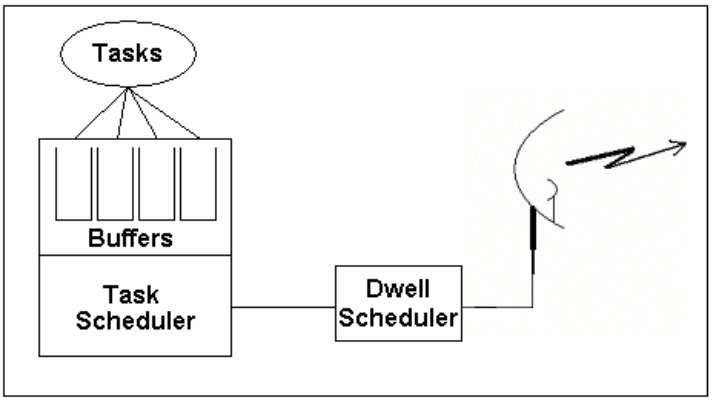

Figure 5.5 Outline of a fuzzy Lyapunov radar scheduler

The resulting scheduler chooses the buffer based on fuzzy Lyapunov synthesis and selects a task from that buffer based on a heuristic function (the part of Eq. (5.1) within the summation). The Lyapunov function (and its derivative) of the scheduler is given in Eq. (5.2).Since the wish is to have bounded buffer sizes, tasks should eventually be executed or deleted, the scheduler should be stable in terms of Eq. 5.2. The MFR can only process one task at a time so only one of the buffer sizes can decrease. Due to fuzzy logic the function can be interpreted as a linguistic function on which the decision rules are based.

$$V = \frac{1}{2}\sum_{j=1}^{4} B_{W\,j}{}^{2}$$
$$\dot{V} \cong \sum_{j=1}^{4}\left(B_{W\,j}\cdot \dot{B}_{W\,j}\right) \qquad (5.2)$$

Note that the notation $\cong$ is used to emphasise that it is a linguistic equation. The rules obtained from these equations result in the choice for a buffer j^* from which tasks are to be processed in each situation. One of the rules is given in table 5.1, where $j^* = -3$ means that buffer 3 should not be chosen. The other rules based on this principle are given in table 5.2.

Table 5.1 First rule deducted from Eq. (5.2)

j*	$\dot{V} \cong (big)\dot{B}_{W1} + (big)\dot{B}_{W2} + (small)\dot{B}_{W3} + (big)\dot{B}_{W4}$
1	*(big)(negative) + (big)(positive) + (small)(positive) + (big)(positive)*
2	*(big)(positive) + (big)(negative) + (small)(positive) + (big)(positive)*
3	*(big)(positive) + (big)(positive) + (small)(negative) + (big)(positive)*
4	*(big)(positive) + (big)(positive) + (small)(positive) + (big)(negative)*
Rule	**If buffer 3 is small and others big, then *j** = -3**

As can be seen in table 5.2, rules use the linguistic expressions ‘small’ and ‘big’. To define these concepts in the system membership functions $\mu_B(B_W)$ and $\mu_S(B_W)$ are used, both satisfying some mild requirements. The first function should be monotonically increasing with $\mu_B(0) = 0$ and $\lim_{B_W \to \infty} \mu_B(B_W) = 1$. The second membership function should be monotonically decreasing with $\mu_S(0) = 1$ and $\lim_{B_W \to \infty} \mu_S(B_W) = 0$.

In [24] it is shown that the use of such membership functions leads to the conclusion that tasks from the buffer with highest B_W value are chosen to be processed. Since the buffer weights have to be bounded for stability, tasks are chosen from the buffer based on their influence on the buffer size.

Table 5.2 Rules for scheduler

Rule	**If**				**Then**
	Buffer 1	**Buffer 2**	**Buffer 3**	**Buffer 4**	***j****
1	*big*	*big*	*big*	*small*	-4
2	*small*	*small*	*small*	*big*	4
3	*big*	*big*	*small*	*big*	-3
4	*small*	*small*	*big*	*small*	3
5	*big*	*small*	*big*	*big*	-2
6	*small*	*big*	*small*	*small*	2
7	*small*	*big*	*big*	*big*	-1
8	*big*	*small*	*small*	*small*	1

Now that Lyapunov synthesis and its effects on the scheduling process are known, the data flow needs to be modelled. The scheduler consists of four separate functionalities as illustrated with data flow charts in figure 5.6. These processes are: adding tasks to the appropriate buffer, calculating buffer weights, deleting expired tasks from the buffers and of course placing tasks in the schedules for the best suited sensor.

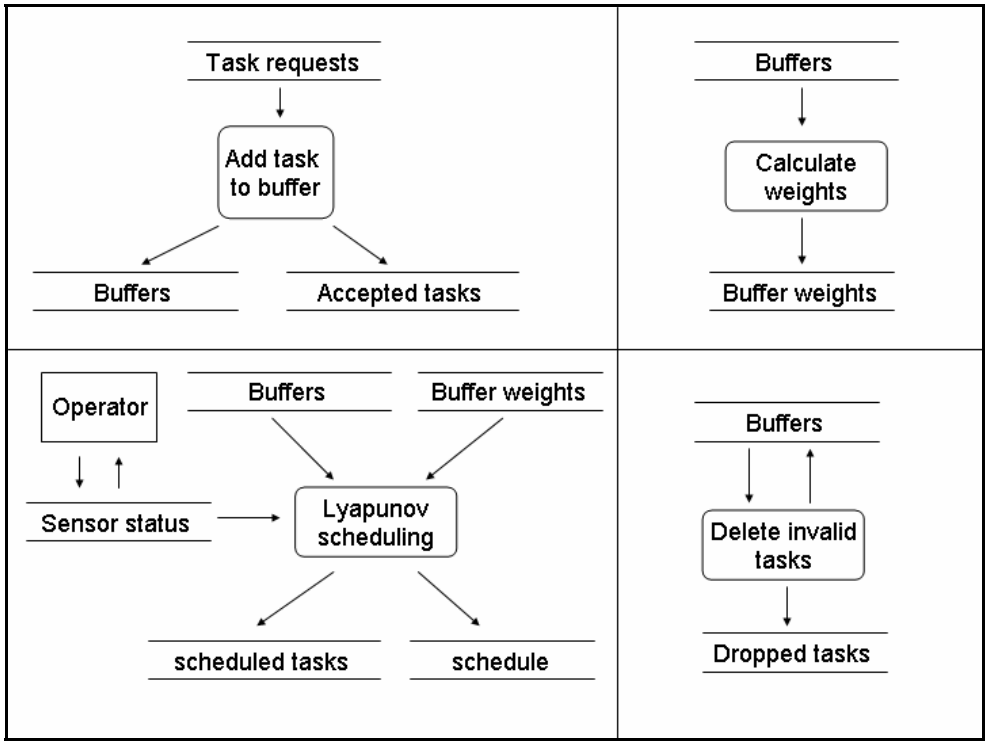

Figure 5.6 The four data processes in Lyapunov based scheduling

Calculating weights and checking validity of tasks are executed without user interference every time step for as long as the system is operating. Placing tasks in the buffers is done whenever a new task requests is given by the CMS.

The most complex function is the scheduling itself which is done on sensor demand. Whenever a sensor becomes available to execute a new task the Lyapunov scheduler assigns the appropriate buffer from which a task is extracted. The status of the sensor is a measure for the ability to execute a particular sensor function, a sensor could be available for executing new tasks, busy with executing a task or not available to the scheduler.

5.4 Genetic neural network

The combined strategy proposed in section 4.8 leads to a scheduling system as illustrated in figure 5.7. In this scheme the incoming task requests are scheduled by a feed-forward NN by grouping *K* tasks and scheduling them like the GA would. Requests are also stored in a taskset. The GA makes schedules based on tasks from the task set. The NN that is used for online scheduling can be retrained at a regular basis. Since the NN is retrained based on recent occurrences the resulting scheduler is adaptive.

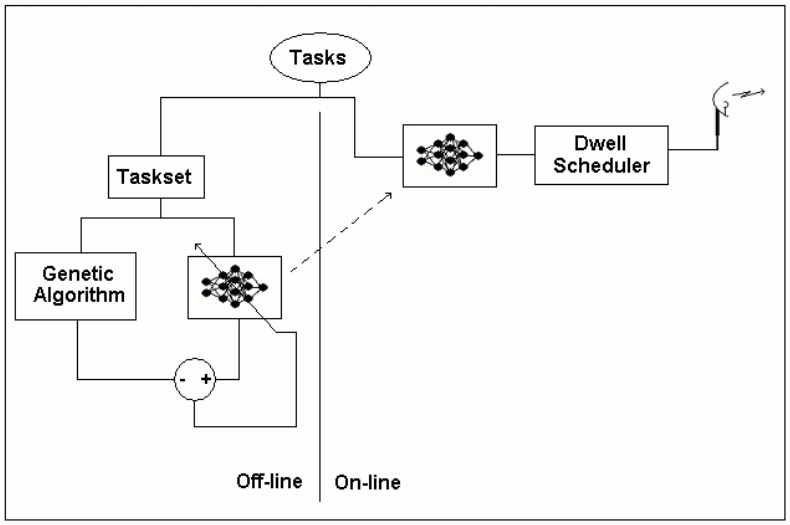

Figure 5.7 Outline of a scheduler using a NN and a GA to update it

Genetic algorithms are often used to find good solutions in search spaces that are too big to be searched exhaustively. Given the size of the search space for scheduling in sensor networks[2] GAs look promising. There are some issues when using a GA in general and one especially when using it in a highly dynamic environment like naval warfare. The first two of these issues are defining a chromosome mapping and defining a fitness function. The issue important for naval warfare is the speed of the algorithm.

[2] E.g. three tasks on the sensors can be scheduled in 24 different ways, four tasks on two sensors in 120 ways.

Although it doesn't take as much time as searching to entire solution space, it is still computational complex, rendering it useless for direct use in sensor scheduling due to the near-real-time characteristic of sensor scheduling.

Fitness function

Before we can use the GA for optimisation the fitness function C_F, Eq (5.3), is defined. This function is maximised to find the optimal schedule. The downside of maximisation is in this case that the maximum is unknown, meaning that when we stop after a certain amount of generations we can't say how (sub) optimal the solution is. The choice for maximisation was made since the goal is to execute as much tasks with high priority on the most suited sensor. A minimisation function could be made (e.g. *K* minus Eq. 5.3) that should go to zero. However, it still couldn't be said where the minimum is at any given time for the optimal solution.

This fitness function places important tasks before less important tasks and takes set-up times into account by trying to minimise the number of switching where this is required. It also tries to position all tasks according to their temporal demands by using a utility. And finally it tries to schedule tasks with a long processing time before shorter tasks. This latter is made less influential by mapping it from $[0,1]$ to $[\alpha,1]$ as can be seen in Eq. (5.1).

Test results will be used to determine values for α, for which of course $0 \le \alpha \le 1$ should hold. The weighing factor κ is used to ensure that the best suited sensor for the task is chosen. How well a sensor can perform a task is dependent of detection probability, classification and identification possibilities and the objects characteristics such as position, speed and manoeuvrability. Since the determination of κ is complex, interviews with experts and operator influence should be used to set this parameter for the various task-sensor combinations.

$$C_F = \sum_{j=1}^{N} \sum_{i=1}^{n_j} \kappa_{j,i} \delta_{j,i} S_{j,i} P_i \left((1-\alpha) \cdot \tau_i + \alpha \right) \tag{5.3}$$

In Eq. (5.3):

C_F	:	Chromosome fitness;
$\kappa_{j,i}$	:	Appropriateness of sensor *j* for task *i*, with $0 \le \kappa_{j,i} \le 1$;
$\delta_{i,j}$	:	Feasibility of task *i* for sensor *j*, 1 if $t_{ct} + t_{td} < t_{dd}$ else 0;
$S_{j,i}$	:	Set-up factor in sensor *j* for task *i*, 1 if no set-up is needed else 0.8;
P_i	:	Priority of tasks *i*, with $0 \le P_i \le 1$;
$\tau_{i,j}$	:	Relative processing time of task *i* on sensor *j*, with $0 \le \tau_{i,j} \le 1$;
α	:	Weight factor to map relative processing time on a different domain, $[0,1] \to [\alpha,1]$, with $0 \le \alpha \le 1$;
n_j	:	Number of tasks for sensor *j*;
N	:	Number of sensors in the network.

Chromosome mapping

Looking at a set of sensors means that different schedules are made. All these schedules have to be combined in a single chromosome, otherwise genetic operators such as cross-over and mutation can't be used. Another reason for mapping these schedules to a single chromosome is that the GA should be able to assign tasks to different sensors. A chromosome mapping was made to facilitate these demands.

Assume three schedules, S_1, S_2 and S_3, for three different sensors. These schedules are made based on a single fixed size buffer and so their concatenation has the same properties as in the single sensor case. The total schedule, $S = \langle S_1 \oplus S_2 \oplus S_3 \rangle$, can therefore still be mapped using the swapping representation (De Jong [10]), denoted by Chr with $S = swapping(Chr)$.

A problem now occurs in determining which part of the schedule/chromosome belongs to which sensor. In this example three elements are added to the chromosome. First element now represents the number of tasks for the first sensor; second element represents the number of tasks for the second sensor and so on.

$$chromosome = \langle 4 \quad 3 \quad 3 \quad C \rangle \rightarrow \begin{cases} S_1 = elements \quad 1 \rightarrow 4 \quad of \quad swapping(Chr) \\ S_2 = elements \quad 5 \rightarrow 7 \quad of \quad swapping(Chr) \\ S_3 = elements \quad 8 \rightarrow 10 \quad of \quad swapping(Chr) \end{cases}$$

This mapping is still incomplete because the first three elements still can't be used in mutation since their sum has to be equal to *K*. This problem is solved by looking at the first elements as weights rather than numbers.

$$chromosome = \langle 8 \quad 6 \quad 6 \quad C \rangle \rightarrow \begin{cases} S_1 = elements \quad 1 \rightarrow a \quad of \quad S \\ S_2 = elements \quad a+1 \rightarrow a+b \quad of \quad S \\ S_3 = elements \quad b+1 \rightarrow a+b+c \quad of \quad S \\ a+b+c \equiv K \end{cases}$$

$$\begin{array}{l} while \quad a+b+c < K \\ \quad add \quad 1 \quad to \quad smallest \quad value \\ end \end{array}$$

$$\rightarrow \quad a = 4 \quad b = 3 \quad c = 3$$

Due to this mapping cross-over and mutation can be used. Since this mapping is highly dependent on the problem domain the construction of new generations should be made to fit the problem.

Note that this is a simple mapping that works. Other mappings, e.g. the mapping used in vehicle routing problem (see [19]), might prove to result in a faster convergence to maximal chromosome fitness. Since the goal of this thesis is to give a conceptual proof of scheduling techniques in sensor scheduling (section 1.3) other possible mappings will not be discussed.

Creating the new population

The new population is generated from the old population with the use of cross-over and mutation. The large part of the new population, 65%, is made by using cross-over on the chromosomes with highest fitness. On this part a mild form of mutation is applied. The second part is made by copying the fittest chromosomes from the old population into the new population. On the fittest chromosomes in this group no mutation is applied. On the others the elements for sensor allocation are mutated.

The final new chromosomes are made by adding new randomly generated chromosomes. These random chromosomes build up 5% of the new population. The reason for these extra chromosomes is to introduce new genetic material in the population and to prevent getting stuck in local optima.

The values for these percentages were empirically determined by tests. Before GAs can be applied for sensor scheduling more tests should be done to optimise the construction of new populations. This however is of course also dependent of the optimal mapping solution.

Training the neural network

Looking at the complexity of the problem and its many representational issues the choice was made to implement the NN ourselves as opposed to using a toolbox. Therefore no interfaces are needed and total control over the network is maintained. Since all algorithms are programmed in Matlab the network was defined using matrices, all in- and outputs are represented by column vectors and the weights between two layers are placed in matrices. The resulting configuration is illustrated by calculating the input for the first hidden layer in Eq. (5.4).

$$Input\_layer1 = \underbrace{\begin{bmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,I} & W_{1,\vartheta} \\ W_{2,1} & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \vdots & \vdots \\ W_{L1,1} & \cdots & \cdots & W_{L1,I} & W_{L1,\vartheta} \end{bmatrix}}_{W_{i\to 1}} \cdot \begin{bmatrix} i(1) \\ i(2) \\ \vdots \\ i(I) \\ \vartheta \end{bmatrix} \tag{5.4}$$

In Eq (5.4):

$Input\_layer1$	:	Column vector with values as presented to the first hidden layer;
$W_{i,j}$	:	Weight between the *i*-th input value and the *j*-th node in the hidden layer;
$i(n)$	:	The input at input node *n*;
ϑ	:	Threshold value, equal to -1;
$L1$	:	Number of nodes in the first hidden layer;
I	:	Number of input nodes.

The output of a node in the hidden layer is calculated with the sigmoid function, shown in figure 5.8. Using this continuous function has the advantage that the derivative can be used in the training process. Updating the weight factors is done with the back propagation algorithm as described in [15].

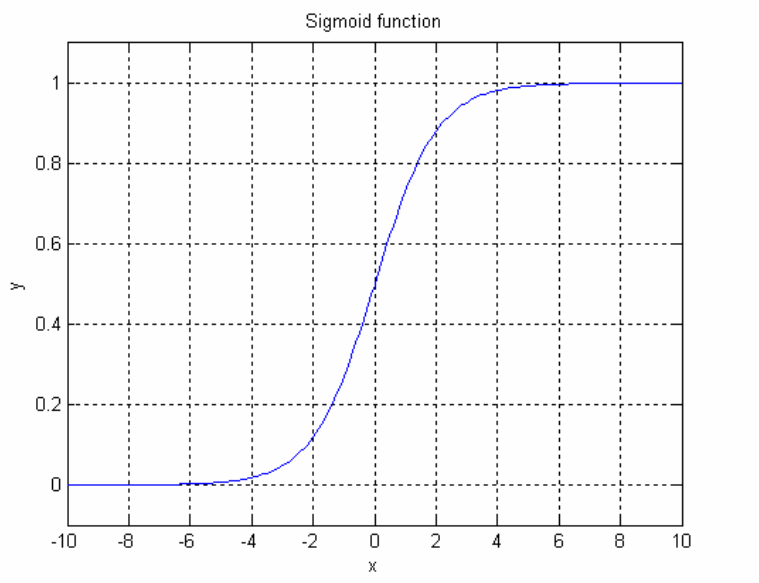

Figure 5.8 Sigmoid function: $y = f(x) = \frac{1}{1+e^{-x}}$

Training of the NN depends on the mode that is chosen by the user. The first option is to use the NN as an identifier for the GA. This would emphasise on combining the optimisation strength of GAs with the speed of NNs. Training a NN like this however proved to be difficult which led to a different approach: using the NN as a scheduling advisor. In this mode the NN recalculates the priority of a task based on the task characteristics. After this re-calculation a simple heuristic is used that schedules tasks solely on this pseudo priority.

5.5 Online use of a GA

Using the GA from section 5.4 for training NNs is not the only option to overcome the issue of speed. Using the GA online can be done by using a buffering scheme similar to the one used in the Lyapunov based scheduler. Here only one buffer with a fixed size K is necessary opposed to using several buffers for Lyapunov. This buffer is filled with default tasks until task requests are received. These are then placed in the buffer instead of a default task. Whenever a sensor becomes available to execute a new task the scheduler looks at the current best solution given by the GA. It identifies the correct schedule for the sensor and only uses the first entry, meaning that only one task is sent to the sensor. Doing so means the buffer only changes slightly, assuming no new tasks were added in the meantime. Since the GA is constantly running this implies that few generations are needed to find a new optimal solution.

In this setting the GA does not have any ending criteria. We therefore can say nothing about the optimality of the schedule that is used. Although this might appear to be a large setback, it most likely is not. Looking at the near-real-time demands on the system in the military domain, a sub-optimal solution on time is better than an optimal solution that is too late.

The outputted schedules however still have to be as good as possible given any amount of time. The maximum fitness in consecutive populations should therefore increase as fast as possible. Some techniques to get this result are: i) injecting heuristic solution into the initial population, ii) make the initial population large enough so all genetic material is available, iii) use parallel processing to speed up the generation process and iv) fine tune the parameters used in the genetic operators.

Scheduling a single sensor is a problem that involves many parameters. Therefore scheduling a network of sensors is even harder. Using GAs for a problem with such a large solutions space is a valid one. The main disadvantage of GAs however is speed. Using the GA as a parallel process is the answer in solving this.

This way of using a GA requires a single buffer. As was the case with fuzzy Lyapunov based scheduling the validity of this buffer needs to be maintained. This leads to three different processes that should be executed parallel to each other:

1. Check buffer for invalid tasks based on due dates;
2. Add new tasks to buffer if buffer has smaller size than K;
3. If a sensor is free, determine which tasks is to be scheduled based on current solution of the GA.

5.6 Hybridisation

The first and possibly easiest possibility for hybridisation was introduced in section 5.5 where the injection of heuristic solutions in the initial population was proposed to increase the online GA performance. To make a hybrid system requires the chromosome mapping to be reversible. Through this reverse mapping the solutions given by heuristics or other schedulers (e.g. the fuzzy Lyapunov based scheduler) can be converted to a chromosome and injected in the initial population.

The GA can then use these solutions and try to improve them. Generation of new populations (section 5.4) ensures the fittest chromosome in the new population to be as least as good as the fittest chromosome in the proceeding population. This means that the hybrid scheduler has a guaranteed performance based on the schedulers used in the hybrid scheduler. Whenever a new sensor becomes available for execution of a task the fittest chromosome is used for the task allocation.

After scheduling a task the buffers change and the schedulers should be used to calculate new schedules that can be injected in the population. This approach will most likely cause a fast conversion to optimal solutions in the real-time setting of the sensor management problem. Parallel processing can be used to execute the different schedulers in order to meet the temporal demands of the hybrid scheduling algorithm.

5.7 Evaluation of chosen models

Evaluation of the proposed models is needed to be able to validate the hypotheses stated in section 1.3. The utility of a schedule is calculated with Eq. (5.5) as introduced by Thaens [21]. In this formula μ_i is defined as $\mu_i = 1$ if task i is scheduled according to its constraints, else $\mu_i = 0$. An optimal scheduler will have utility 1, meaning all tasks are properly scheduled.

$$U = \frac{\sum_{\forall i} \mu_i P_i}{\sum_{\forall i} P_i} \tag{5.5}$$

The utility measure is used in three different ways. First of all the overall utility is used. Secondly, the utility can be calculated and plotted for time intervals, giving an indication of the course of the utility in time and finally of the minima and maxima of the utility in time.

Another operational demand is that the system keeps high performance when many targets exist in the environment. Although each algorithm has a saturation point, the point where even high priority tasks have to be dropped, the models suggested here need to be tested on their behaviour around that point. Of course also on when that point is reached, when saturation occurs too soon the model is less suited for integration in combat systems.

Another important factor at the point of saturation is which tasks are being dropped. Which function types receive extra priority is mission dependent. In general, and as criteria in this thesis, it can be said that missile guidance tasks should be dropped less than search and track tasks. The evaluation feature is calculated similar to the utility, by dividing the number of executed tasks by the number of requested tasks for each sensor function type. The total list of criteria is given in table 5.3.

Table 5.3 Evaluation criteria for schedulers

Principle	Criteria
Utility	$U_{overall}$;
Utility difference in time	$U_{\max} - U_{\min}$;
Percentage of executed search tasks	$\frac{\#\ executed\ \ search\ \ tasks}{\#\ requested\ \ search\ \ tasks}$;
Percentage of executed track tasks	$\frac{\#\ executed\ \ track\ \ tasks}{\#\ requested\ \ track\ \ tasks}$;
Percentage of executed MG tasks	$\frac{\#\ executed\ \ MG\ \ tasks}{\#\ requested\ \ MG\ \ tasks}$;
Percentage of executed TI tasks	$\frac{\#\ executed\ \ TI\ \ tasks}{\#\ requested\ \ TI\ \ tasks}$.

6 Implementation

This chapter discusses the implementation of the models introduced in chapter five. All models were implemented using Matlab. All programs that were developed during this thesis work can be found on the CD-ROM that comes with this report. Section 6.1 describes the representation of task requests which is used in all developed schedulers. The next three sections describe implementation details of the schedulers. In the final section, 6.5, the implemented evaluation tools are discussed.

6.1 Modelling tasks

Based on the notion in section 5.2 about task definition tasks can be represented by row vectors containing the four mentioned elements: task type, priority, due date and duration. The two last elements are time indication and can be represented in either simulated time, real time or system time steps. Priority has a value between zero and one and can be generated randomly in Matlab or based on some more operational information.

The only remaining element to be defined is the sensor function type indication. For implementation ease the choice was made to represented the different types with numbers as shown in table 6.1.

Table 6.1 Representation of sensor function types

Sensor function type	Representation number
Search	1
Track	2
MG	3
TI	4

6.2 Fuzzy Lyapunov based scheduler

The data model of this scheduling methodology was discussed in section 5.3. Based on the data flow chart Matlab files were created that perform the necessary actions. Input of the scheduler is a task request made by the CMS. The output is the schedule for the sensor under consideration. In the design all functions were meant to work parallel, in running the simulation however the functions are executed sequentially. The functions, variables, in- and output are shown in figure 6.1 for the Lyapunov based scheduler.

In figure 6.1 can be seen that there are three types of functions. The first type of functions is the '*call*' function. This indicates a function that is activated by the operator or the CMS. The 'add' function is executed when an input is received and the 'evaluation' function is activated when the performance of the scheduler needs to be checked by the system or the operator.

The second type of functions is the '*time*' function, which are executed each time step. The function 'del_task' ensures that all requests are still valid given the current time. Weights of the buffers are time dependant and should therefore be updated every time step, this is done by the function 'weights'. The last '*time*' function is 'timeline'. Every time step it checks whether the sensor is available for new tasks, if it is it calls on 'lyapunov' to select a new task to be executed from the buffer.

With the 'lyapunov' function the third type of functions is addressed, the '*hidden*' functions. These work in the background and are not accessible for the operator or the CMS. The only way they can be activated is through other functions. The first, 'lyapunov' selects the next task to be executed from the buffer based on the current values for the buffer weights. The 'bufferp' function creates and outputs plots when the 'evaluation' functions calls on it.

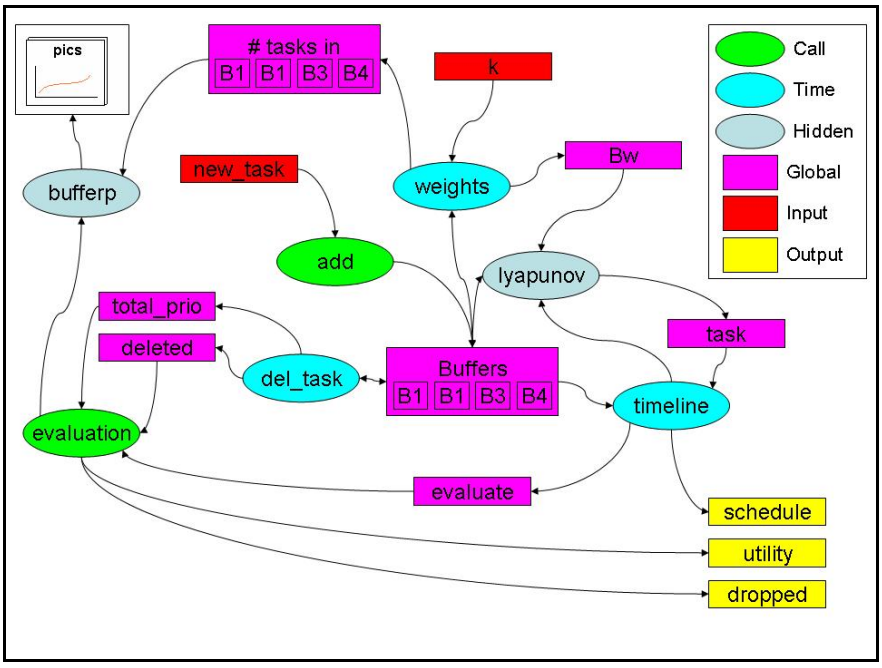

Figure 6.1 Functions and variables in the Lyapunov based scheduler

During runtime of the implemented scheduler relevant parameters are maintained in order to keep track of the performance of the scheduler, the performance criteria are described in section 5.7. When adding tasks to the buffer a separate store is maintained with all requested tasks. All scheduled and deleted tasks are also stored in separate data structures. This means that the utility of the scheduler can be monitored during run-time giving the operator the chance to keep track of the system and if necessary change parameters in order to increase performance.

In section 5.3, Eq. (5.1) was used to calculate the weights for different buffers. In this equation two parameters were introduced that needed fine tuning, namely α and γ. In order to look at their influence on the utility a random[1] task request list with 700 inputs was made for four function types and one MFR. Results of this scheduler with different values for α and γ are shown in figures 6.2 and 6.3. Based on these results values for α and γ were chosen to maximise utility and minimise the number of not-executed tasks.

[1] After the random generation of task request they were placed in order of due date

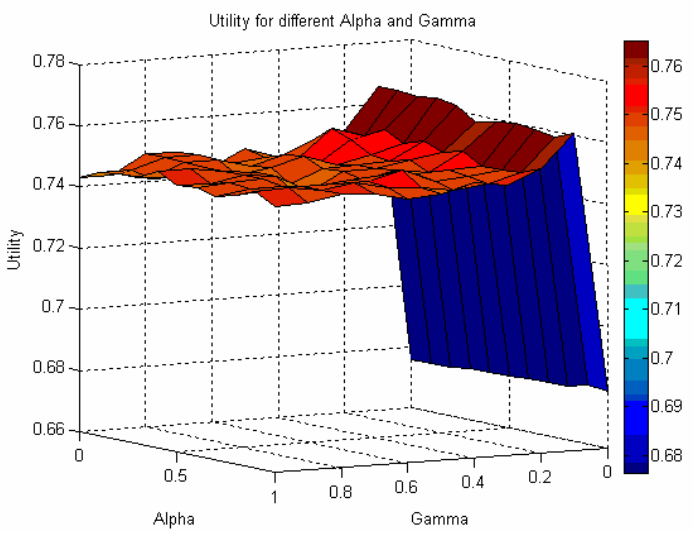

Figure 6.2 Utility of the schedule for 700 random task requests

Figure 6.2 shows that the influence of both α and γ are very small. If this is really the case it remains to be seen when more realistic request lists are given to the scheduler. An explanation for this could be the randomness of the request list. The peak in utility is however remarkable based on the previous statement and confirms the assumption that the weighted task duration is a factor of importance in the scheduling of tasks.

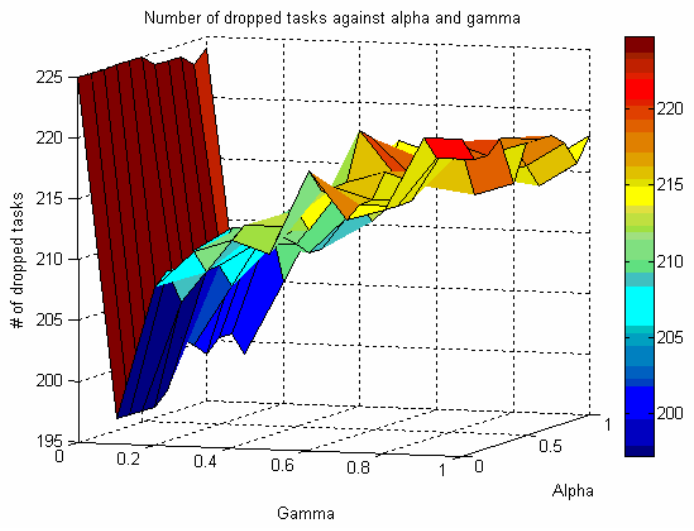

Figure 6.3 Number of dropped tasks in scheduling 700 random tasks

In figures 6.2 and 6.3 the apparent relation between the number of dropped tasks and the utility can be seen. Dropping more tasks in general means obtaining a lower or at maximum equal utility. Both figures also show an interesting behaviour for $0 < \gamma < 0.3$. Figure 6.4 therefore shows the utility for this region. This plot shows that the choice for the parameters should be: $0.7 < \alpha < 1$ and $0.1 < \gamma < 0.2$.

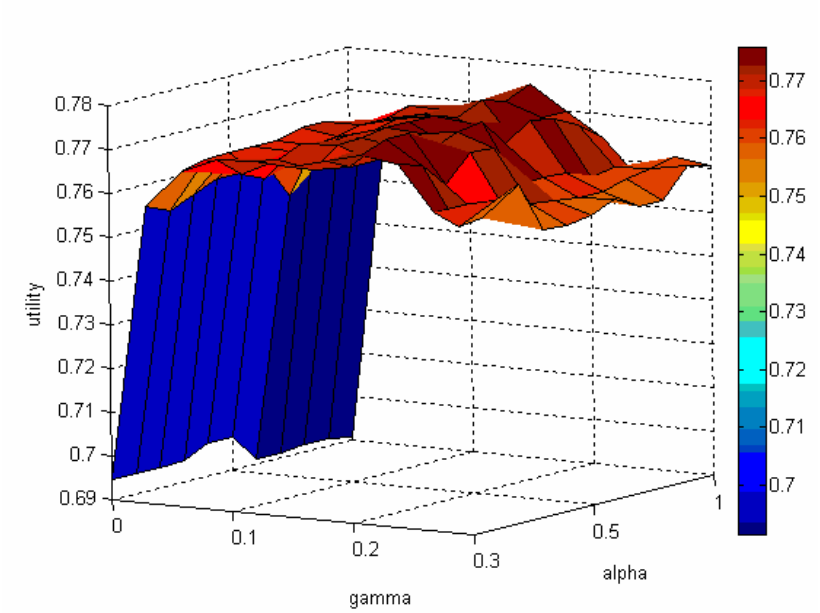

Figure 6.4 Utility for 700 random task requests for small γ

After choosing values for α and γ, the k-factor for each buffer should also be determined. Looking at the problem of emphasising the importance of MG and TI the values for these two buffer are considered in tests, search and track buffers are given a k-factor 1. Figure 6.5 illustrates the influence of *k* for MG and TI tasks on the resulting utility.

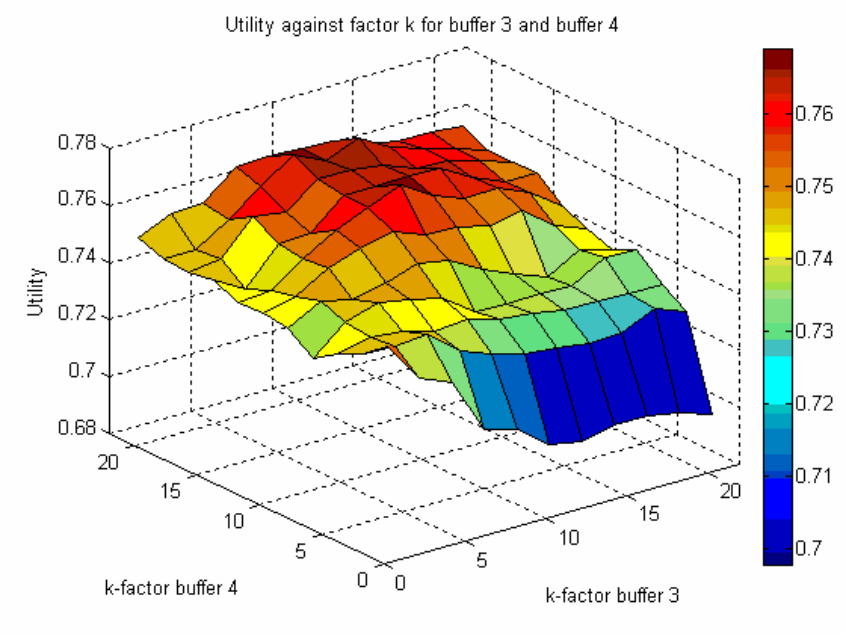

Figure 6.5 Utility for 700 random tasks for different values of k

6.3 Genetic neural network scheduler

The basic model of the GNN was given in section 5.4 along with the principle formula for calculating the chromosome fitness. This section will emphasise on the implementation of the actual scheduler. More specifically on the part that trains the NN, since running a feed-forward NN is relatively simple. Figure 6.6 shows the architecture of the scheduler with some important variables. The entire learning process is implemented as a hidden function so the user has no direct influence on this process. Only by setting the learning parameters and NN sizes can the user influence the scheduler. The two main functions, running the GA and training the NN, are discussed separately.

This algorithm is only implemented for a single sensor; the GA is therefore relatively simple. Task request are placed in a fixed size buffer for which a schedule is made. The schedule consists of integer values, each pointing to a task in the buffer. By using such a schedule each integer can only appear once in a schedule. It therefore resembles the travelling salesmen problem, [10], for which chromosome mappings have already been developed. Two of these mappings, ordinal and swapping representation (De Jong [10]), were implemented and compared. Theory on GAs can be found in [14]. To illustrate the use of chromosome mappings in the single sensor GA situation the code for the swapping representation is given in figure 6.7.

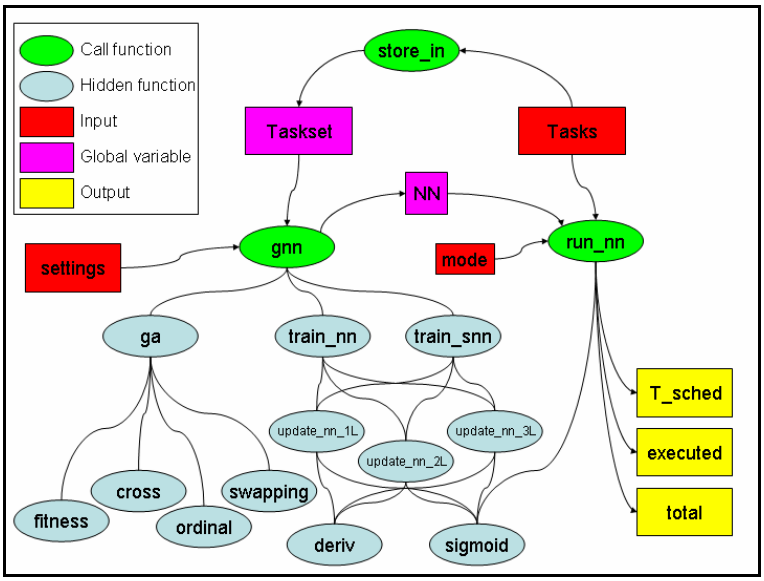

Figure 6.6 Software architecture of the Genetic-Neural Network

The choice was made to implement all the functions without using existing software. This allows for total control of all the parameters and no interfaces had to be made to have communication between different functions. Downside of course is the time needed for the implementation. The feed-forward NN e.g. can be chosen with up to three hidden layers. For each of the possible number of hidden layers a weight update function is implemented. The latter of course is the backpropagation algorithm described in [15]. Since the modelling was based on matrix representations this could be done with relative ease.

```
function [R]=swapping(S)
K=length(S);
R=[1:K]; % set default route
x = 1;
while x <= K
      % make sure that gene-value does not exceed K^2-1
      if S(x) >= K.^2 - 1
         S(x) = mod(S(x), (K.^2));
      end
      A=floor(S(x)./K)+1; B=S(x) - (A-1).*K +1;
      C=R(A); D=R(B); R(A)=D; R(B)=C; %first extract and then replace
      x = x+1;
end
```

Figure 6.7 Matlab code for 'swapping' function

The results of the GA depend on several parameters that need to be set to optimise results. First parameter to be optimised is introduced in section 5.4 in the fitness function. As was the case with Lyapunov the weighing factor for task duration needs to be determined. Executing the GA ten times on a random generated task set of 50 tasks gave the result showed in figures 6.8 - 6.10. Results here confirm that a proper mapping of the problem to chromosomes can avoid the GA from getting stuck in local optima.

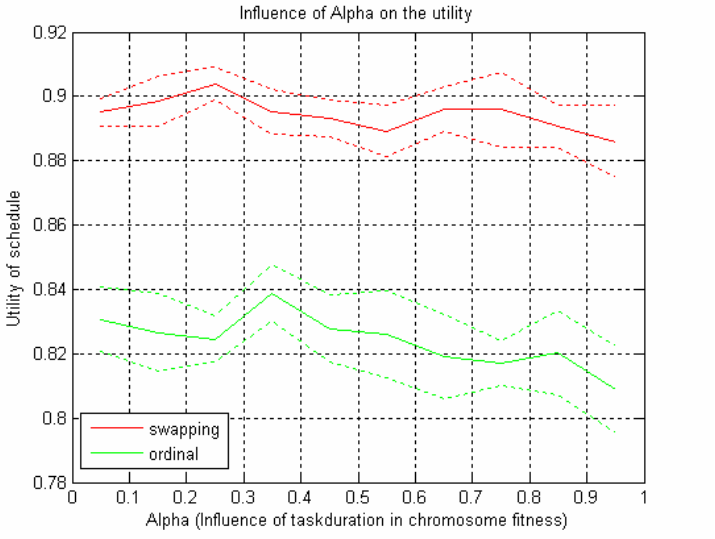

Figure 6.8 The influence of α on the utility of the schedule for two different chromosome mappings

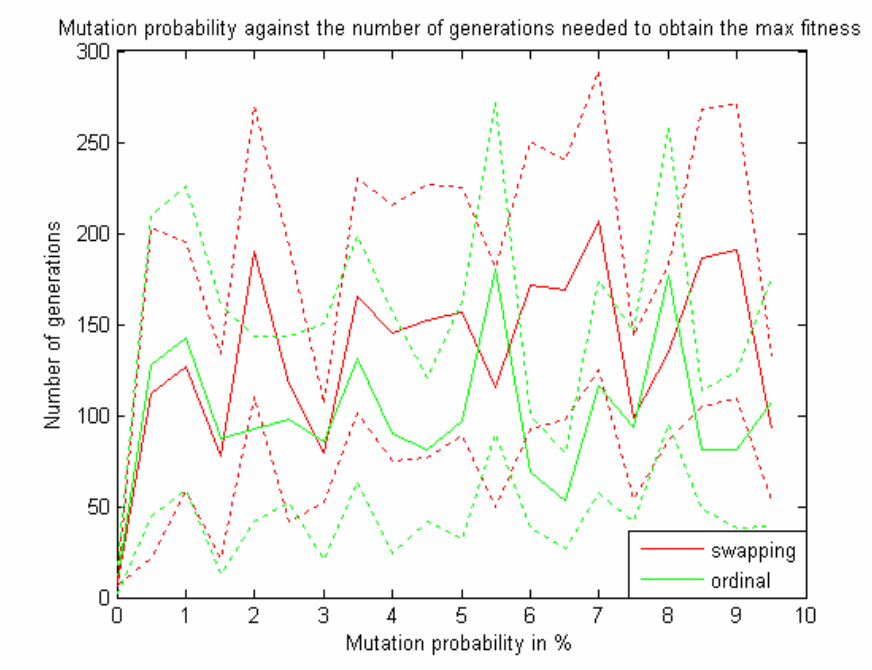

Figure 6.9 Generations needed for maximum fitness for chromosome length 10 over ten runs

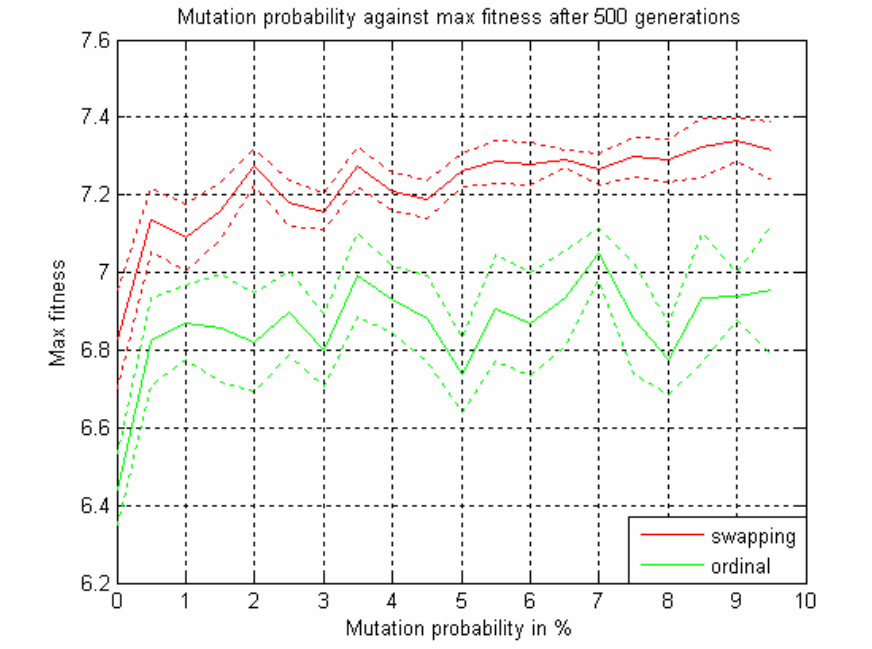

Figure 6.10 Maximum fitness after 500 generations
over ten runs for chromosome length 10

In all cases however the problem proved to be too complex for a good generalisation by the NN. Figures 6.11 and 6.12 illustrate this based on the error development on a test set during the training of the NN. Besides the complexity of the problem, the reason for the high error could be caused by i) improper and/or too few training data or ii) wrong implementation of the trainings algorithm. Figure 6.6 shows that different configurations of the network were possible with a maximum of three hidden layers. This is shown by the three different programs to update the weights in the NN. Theoretically this means that the size of the remaining error is not likely to be caused by too few or too many hidden layers.

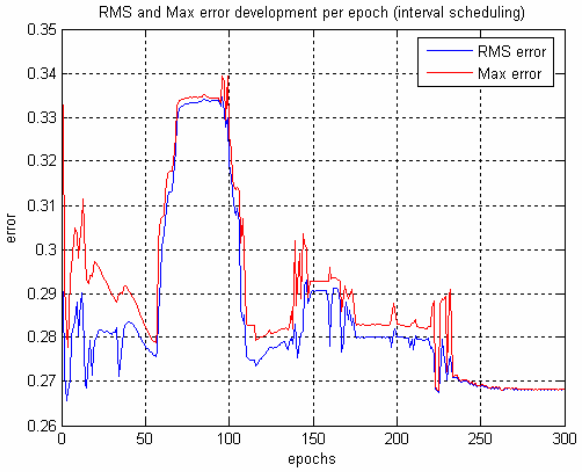

Figure 6.11 Error development in scheduling intervals

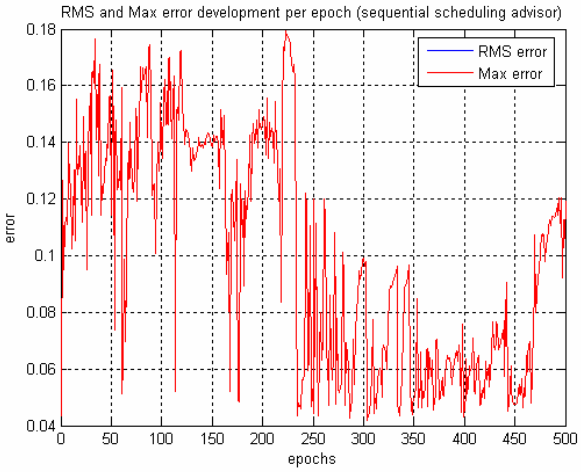

Figure 6.12 Error development in the scheduling advisor

Some first conclusions about the use of this scheduling method are: i) use the NN in the mode of scheduling advisor and ii) use the swapping representation for the GA. The downside of using NNs is illustrated by figures 6.11 and 6.12, the first figure shows that the error of the interval scheduler stays above 20%. In the scheduling advisor the error is lower but overtraining might be an issue here. After 450 epochs the error starts growing. Further, the minimisation of the error does not seem to drop steadily in time so no guarantees can be given about the quality of the resulting network.

6.4 Online use of GA

The online GA is implemented according to the program flow chart of figure 6.13. Some choices however needed to be made since the model is based on parallel processing. The actual tests have to be executed in a sequential setting. An assumption now has to be made about how many generations the GA can perform between two task allocations.

Here the choice was made to use a random number with a maximum of 50. This number can be reached and exceeded in actual applications when dedicated parallel processors work on the generations of the GA. A random number was chosen to simulate the effect that the number of generations between choosing two tasks is unknown in actual systems. Another reason is to show that the GA does not need many generations to find good solutions in the online setting.

This implementation uses basically the same GA as the GNN scheduler. Figure 6.7 showed only the swapping representation which is used in the single sensor case. When using multiple sensors the chromosome mapping of section 5.4 needs to be implemented. This mapping is coded as shown in figure 6.14.

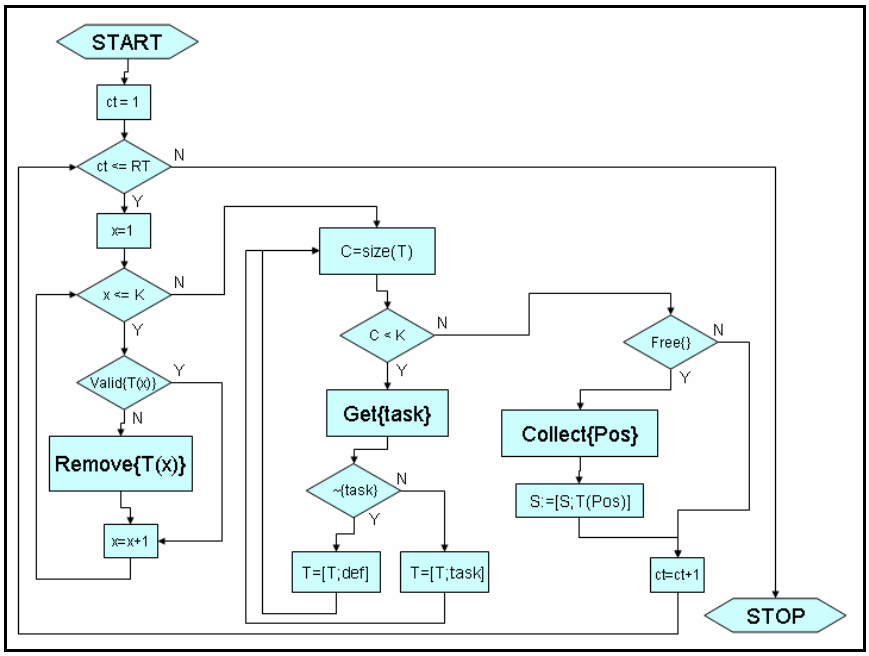

Figure 6.13 Program flow chart of sequential implemented on-line use of GA in scheduling

6.5 Hybrid scheduling

Section 5.6 introduced a hybrid scheduling scheme that uses the online GA configuration. Implementing such a hybrid scheduler means that the basic outline from the online GA stays the same. Some minor adjustments were made in the fuzzy Lyapunov based scheduler in order to use its output in the hybrid setting. The output of a first-in-first-out scheduler was also included in this hybrid scheduler.

The other problem, also discussed in section 5.5, is reversing the mapping function. The swapping representation is used to map chromosomes to schedules. In the hybrid scheduler the schedules from the different schedulers need to be converted to chromosomes so they can be injected in the population.

The chromosomes (section 5.4) were constructed in a way that the part of sensor allocation can be exactly the same in the schedule and in the chromosome. Reversing the 'swapping' function, figure 6.7, is therefore the only problem. The Matlab code for this reversal is given in figure 6.15. With this function the schedules can be converted to chromosomes. The entire implementation of the hybrid scheduler is now done by using the earlier developed code and combining it with each other.

6.6 Evaluation tools

After the implementation of the actual schedulers some more functions have to be made in order to compare the scheduling algorithms based on the same principles described in section 5.6. All evaluation parameters can be obtained by comparing the outputted schedule with the inputted requests.

This of course means that all schedulers use the same evaluation function, which therefore only has to be implemented once. Functions are made to plot histograms of requested, scheduled and dropped tasks. Another plot can be made by monitoring the utility for intervals of one second.

All schedulers use the same input so this is evaluated only once by a separate function. The evaluation of the output is done by using the input globally and comparing it with the outputted schedule.

```
function schedules = mapping(chromosomes)

% Determine population size N
N = size(chromosomes,1);
% Determine number of sensors
Num_Sen = size(chromosomes{1}{1},2);
% Determine number of tasks
Num_Tasks = size(chromosomes{1}{2},2);

% Declare schedules to be a cell array
schedules={};

% Do for all chromosomes in the population
counter1 = 1;
while counter1 <= N
      % Make sure that number of tasks per sensor
      % in total is correct
      A = chromosomes{counter1}{1};
      A = [A;1:length(A)];
      A = sortrows(A',1)';
      factor = floor(Num_Tasks.*A(1,:)./sum(A(1,:)));
      counter2 = 1;
      check = sum(factor);

      while counter2 <= (Num_Tasks - check)
         factor(counter2) = factor(counter2) + 1;
         counter2 = counter2 + 1;
      end

      A(1,:)=factor;
      clear factor check
      counter2 = 1;

      while counter2 <= Num_Sen
         Position = A(2,counter2);
         schedules{counter1,1}{1,1}(counter2) = A(1,Position);
         counter2 = counter2 + 1;
      end
      clear Position A counter2

      % For schedule itself use swapping representation
      schedules{counter1,1}{2,1}=swapping(chromosomes{counter1,1}{2,1});

      counter1 = counter1 + 1;
end
```

Figure 6.14 Function for the chromosome mapping in a multi sensor situation

```
function S = de_swapping(schedule)

K = length(schedule);

S = zeros(1,K);

counter1 = 1;
while counter1 <= K
      % Find the needed value in current schedule
      pos_c1 = schedule == counter1;
      pos_c1 = [pos_c1' [1:K]'];
      n = 1;
      while pos_c1(n,1) == 0
            n = n+1;
      end

      B = n;
      A = counter1;

      if A==B
            S(K-counter1+1) = 0;
      else
            S(K-counter1+1) = B + (A-1).*K - 1;
            C=schedule(A); D=schedule(B);
            schedule(B)=C; schedule(A)=D;
            clear A B C D
      end

      counter1 = counter1 + 1;
end
```

Figure 6.15 Reverse swapping function

7 Results

After the implementation of the different schedulers they are compared based on a simulation. The simulation is discussed in the first section of this chapter and how task requests were made based on this simulation is discussed in the second section. The schedulers are not only compared with each other but also with the commonly used first-in-first-out (FIFO) scheduler, discussed in section 7.3. The results are finally discussed in section 7.4.

7.1 Simulation

In order to compare schedulers they need to be tested in overload situations. To reach such an overload situation the requested sensor time must exceed to available sensor time. The basis for the used task durations can be found in section 2.3. The duration of LVS was decreased, with respect to the duration determined in section 2.3, in order to request more of these tasks. This choice is supported by assuming a smaller search area for the search function and/or at a shorter range.

Another assumption was made for the generation of tasks. In the model of section 5.1 the scheduling process is based on the object store that generates tasks based on the (uncertainties on) objects characteristics. To simplify the task generation process in the simulation, update rates were chosen for the different sensor functions. The necessary set-up time was also chosen to increase scheduling difficulty and was set to 0.01 s.

The simulation is based on the four sensor functions described in section 3.3 and the simulated time is one minute. Each second a horizon search is requested (update rate 1 Hz) with task duration 0.12 seconds. Other search functions are requested by four limited volume searches (LVS) (each with an update rate 2 Hz and task duration 0.04 seconds). Two of these objects requesting LVS are inbound, meaning their priorities continuously increase during the 1 minute simulation. One other is an outbound weapon carrier (priority decreases) and the last object is initially inbound but reaches the closest point of approach after 14 seconds where after it follows an outbound course.

During the simulation three tracks are maintained each with update rate 5 Hz and a task duration of 0.03 seconds. The first is a neutral air contact that is inbound for the first 14 seconds. The second track is an inbound hostile object resulting in a high priority track. The last track is a friendly surface contact with constant priority.

The combination of all these tasks does not yet ensure a saturation of the scheduler. To obtain an overload some missile tracks (18 in total) are added to the simulation. These missiles require countermeasures so MG and TI tasks are also added to the requested tasks. The table containing all objects and their resulting priorities in time can be found in appendix B.

7.2 Task requests

The simulation environment described in the previous section should now be written as a set of task requests. For each task request the format, described in section 6.1, is implemented as a row vector. The entire task set of requests can therefore be represented in a matrix.

To obtain the task set the table from appendix B was imported to Matlab as variable ‘A’. In Matlab an ‘m-script’ was then created to convert the objects priorities to a task set. To illustrate the principle behind this conversion, the conversion of the second column of ‘A’ to task requests will be discussed in more detail.

In figure 7.1 the code is given to make task requests based on the second column. First the priorities in time are placed in a column vector. In front of this column a column filled with the number ‘1’ is placed, indicating a search task (table 6.1). The resulting matrix is then copied and the two are placed below each other. This is done because the update rate is 2 Hz, in each second two task requests are generated. On the third column the due date are placed, each task request has a deadline half of a second later than the proceeding request. On the final column the task duration is placed, being 0.04 s.

```
% Limited Volume Search 1: 2 Hz (column 2)
TaskSet_LVS1(:,1) = A(:,2);
TaskSet_LVS1 = [ones(60,1) TaskSet_LVS1];
TaskSet_LVS1 = [TaskSet_LVS1;TaskSet_LVS1];
TaskSet_LVS1 = [TaskSet_LVS1 [0.5:0.5:60]' 0.04.*ones(120,1)];
```

Figure 7.1 Code to make task requests out of column two of the object-table

Using the same principle for each column a task set was constructed, requesting 2,458 tasks in the simulated second. Another assumption was made on when the tasks are requested. For simulation ease tasks that need to be executed in a certain second are requested at the beginning that particular second. A histogram was made representing the task requests per second and is show in figure 7.2.

In this figure the scheduling problem is apparent for the single sensor situation. Each second means a second of execution time in which also a set-up might be necessary between TI and other tasks. Since the requested time exceeds the available time, tasks have to be dropped.

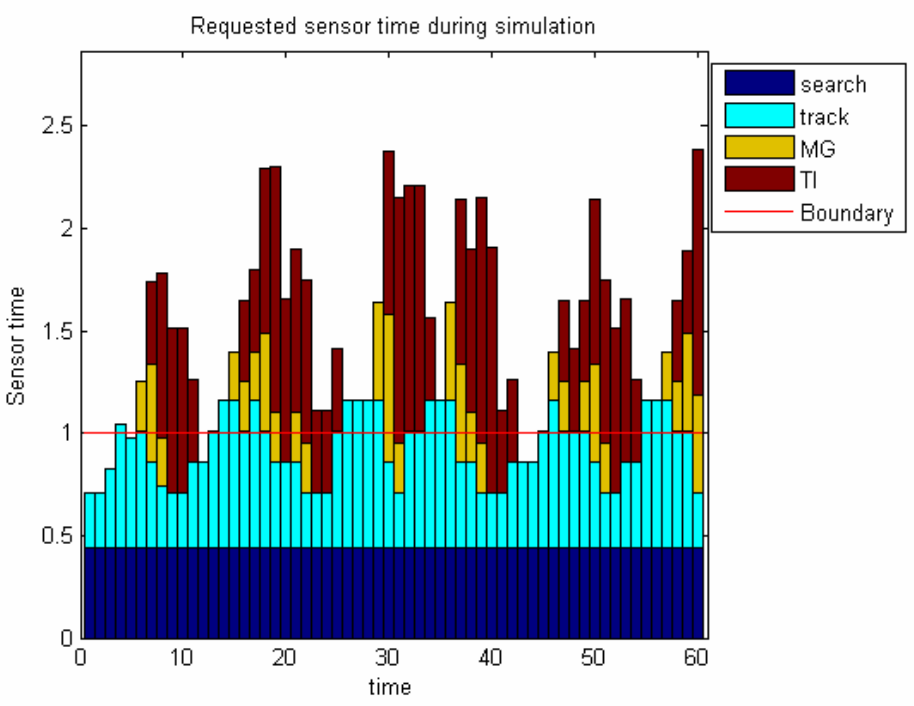

Figure 7.2 Histogram of the requested tasks during simulation

7.3 First-In-First-Out

The FIFO-scheduler was implemented to avoid that the schedulers developed in this thesis could only be compared with other. The principles of this scheduler are easy: simply queue the task requests based on the order in which they were requested. Since all tasks that have to be executed during a certain second are requested at the beginning of that second the results of the FIFO scheduler might improve in actual implementation.

The results, as illustrated in figure 7.3 and table 7.1, of this scheduler give rise to the expectation that improvements can be made. Table 7.1 shows that half of all requested TI tasks are executed whereas 70% of all search tasks are executed. In terms of utility this means that it could already be increased by dropping more search tasks in favor of TI tasks since the latter have a higher priority.

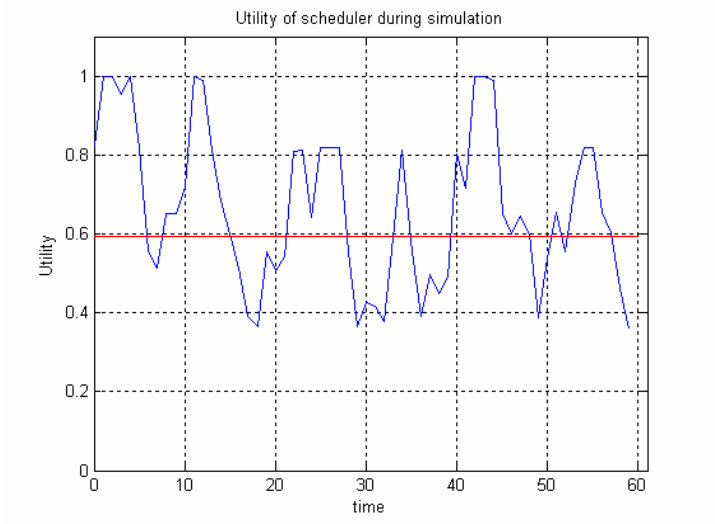

Figure 7.3 Utility in time using the FIFO scheduler

Another improvement is purely based on the operational setting of the problem. TI tasks are executed so deployed weapons will be guided towards a target thus reducing the risk to the mission. Dropping these tasks in order to perform search and track tasks will generate more task requests since surrounding hostile objects are not destroyed and it is likely that own assets will be destroyed. These newly generated track tasks then cause more TI and MG tasks to be dropped and more deployed weapons will miss their target.

Table 7.1 Results of the FIFO scheduler

$U_{overall}$	*0.5938*
$U_{\max} - U_{\min}$	*0.6437*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ of search tasks	*0.711*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ of track tasks	*0.615*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ of MG tasks	*0.507*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ of TI tasks	*0.502*

7.4 Results of the three developed schedulers

Using the generated task set as benchmark all three developed schedulers are used to schedule the requested tasks in time. For each scheduler the results are represented in the same way. Three plots, one illustrating the utility in time, one showing a histogram of the tasks that were dropped and a histogram representing sensor time allocation. The evaluation parameters are used to compare the schedulers in more detail in section four.

Utilities

The utility in time of the three schedulers is shown in figures 7.4 – 7.6. Based on the overall utility the fuzzy Lyapunov based scheduler has the best performance. This however isn't the only criterion. To maximise to sensor usage the minimum utility should be as high as possible. The online use of the GA has the highest minimum value of all four schedulers. This means that this scheduler makes good choices in overload situations.

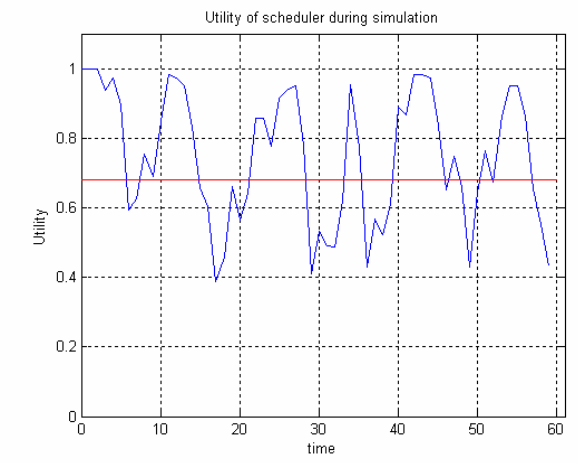

Figure 7.4 Utility of Lyapunov scheduler

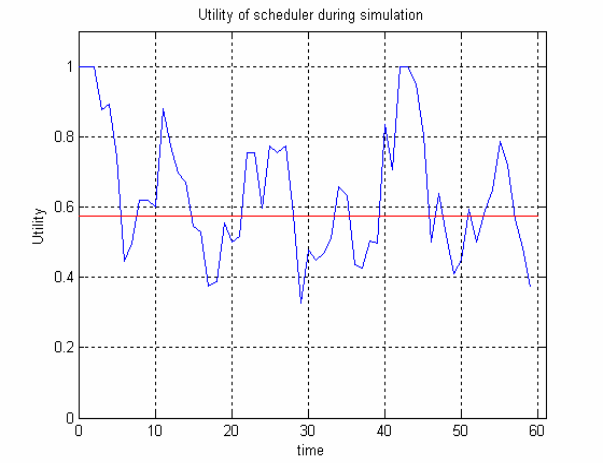

Figure 7.5 Utility of GNN scheduler

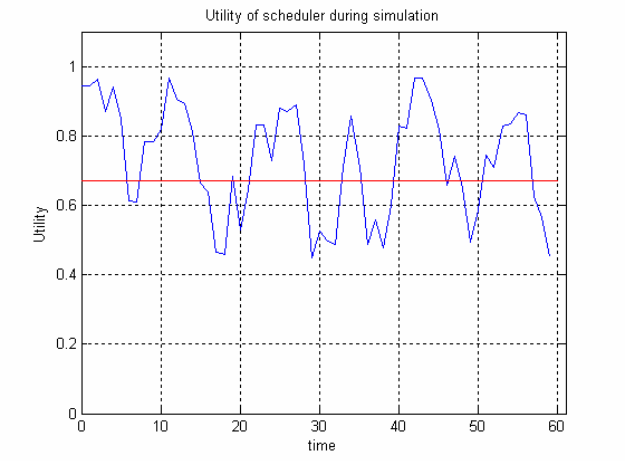

Figure 7.6 Utility of online GA scheduler

A downside of the online GA is the fact that it never found 'perfect' schedules (utility equals 1) whereas other schedulers did. This might be prevented by injecting heuristic solutions (and the Lyapunov solution) in the initial population or by using more computer power allowing more generations between sensor availability.

The fuzzy Lyapunov and the online GA schedulers outperform the FIFO scheduler (figure 7.3) in this simulation. The GNN scheduler's results are disappointing since the FIFO scheduler performs better.

Dropped tasks

The histograms of the dropped tasks are shown in figures 7.7 – 7.9 and show some important results of the schedulers for their use in military applications. These histograms give an indication on how the different schedulers perform in situations where task requests need to be dropped. Minimising the risk during the mission implies that when a weapon is deployed it should hit the target so weapon guidance tasks should not be dropped easily in the scheduling process.

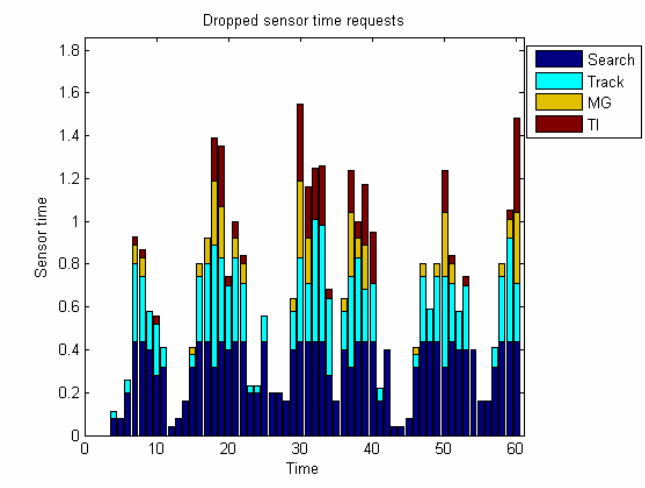

Figure 7.7 Dropped tasks with Lyapunov scheduling

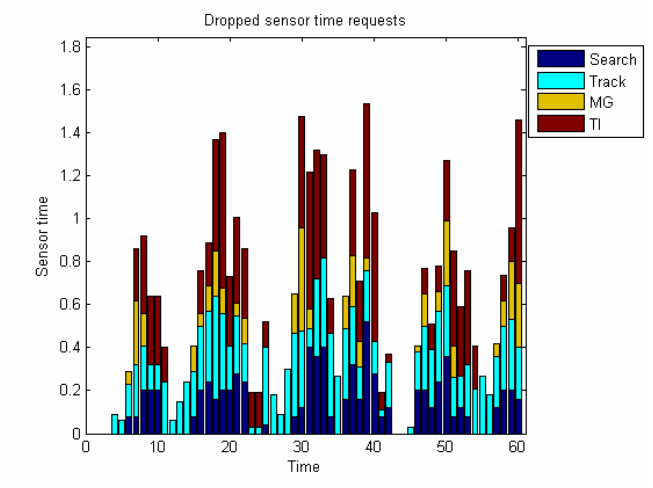

Figure 7.8 Dropped tasks with GNN scheduling

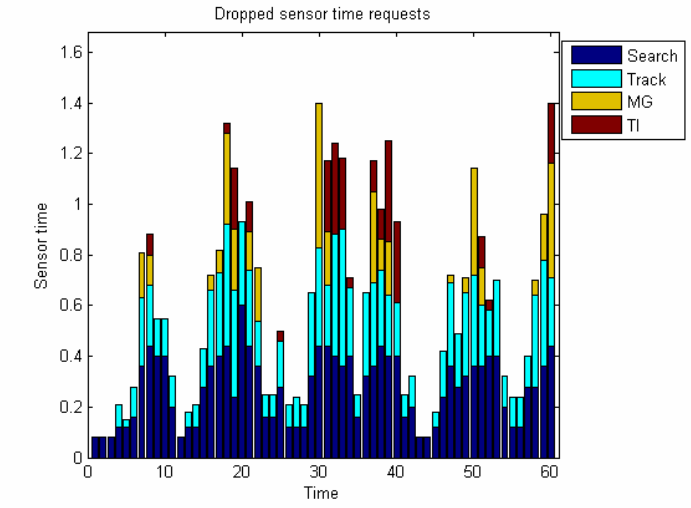

Figure 7.9 Dropped tasks with online GA scheduling

Figure 7.2 indicates that several of these guidance tasks will have to be dropped during the simulation. Comparing figures 7.7 – 7.9 gives an indication how well the different schedulers deal with overload situations. Obvious is that the GNN drops many guidance tasks, both TI and MG, in favour of search tasks which it drops the least. This is unacceptable in the military application. The reason for this is probably due to training difficulties since the online GA doesn't have this problem at all. This online GA scheduler has best performance when looking at the number of dropped TI tasks.

Figure 7.6 showed that the online GA didn't obtain the maximum utility of one, also illustrated by the histogram of dropped tasks. Where the fuzzy Lyapunov based, the GNN and the FIFO scheduler drop no tasks at all in the first couple of seconds, the online GA already drops some search tasks.

This decreased performance of the online GA scheduler can be explained by looking at the implementation. Figure 6.8 showed that the variance in the number of needed generations is relatively high, in average the needed number is at least 50. Since the number of generations executed in the sequential implementation of the scheduler is randomly chosen between zero and 50 it is very likely that the schedule is far from optimal when tasks are allocated in the first couple of seconds. However, though not yet optimal the results still look promising.

Hybridisation of the GA with a heuristic like FIFO or even with the fuzzy Lyapunov scheduler is likely to solve this problem. Moreover, it will probably lead to a fast scheduling algorithm with a predictable minimum performance. Both of these characteristics are important with respect to the military aspect of the scheduling domain.

Resulting sensor use

The used sensor time is illustrated by means of histograms. The used time per sensor function is shown per second as well as the required set-up times. For the three schedulers these histograms are shown in figures 7.10 – 7.11. Due to some rounding problems during the visualisation process it looks like the used sensor time exceeds the available time. This however is only caused by rounding, the exceeding time is compensated in the proceeding or following second.

These figures show interesting results. The reason that the minimum utility is smallest with the online GA can be seen in these figures. The set-up times required for the execution of the tasks is smallest for this scheduler. This explains that the minimum performance of the online GA is higher than that of the fuzzy Lyapunov scheduler. An improvement to the fuzzy Lyapunov scheduler could be made by adding a factor in the formulas for calculating buffer weights that penalises a switch between buffers that requires a set-up time.

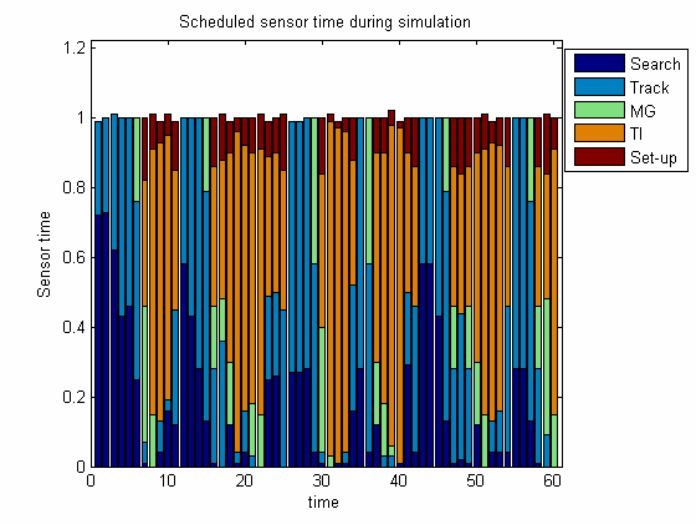

Figure 7.10 Sensor use with Lyapunov scheduling

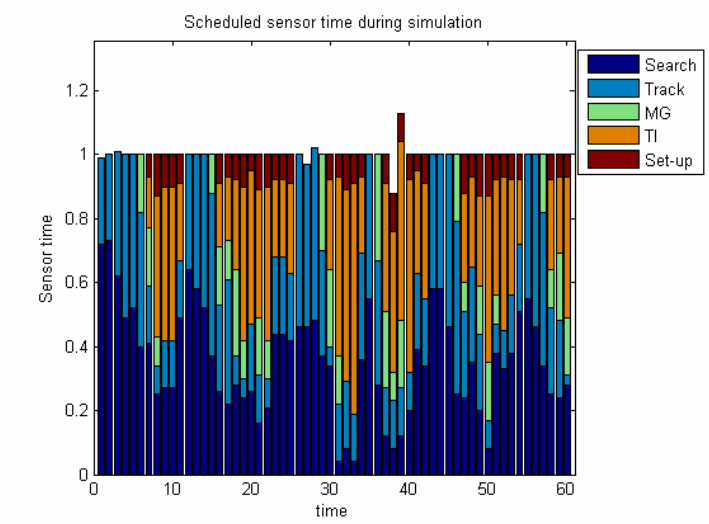

Figure 7.11 Sensor use with GNN scheduling

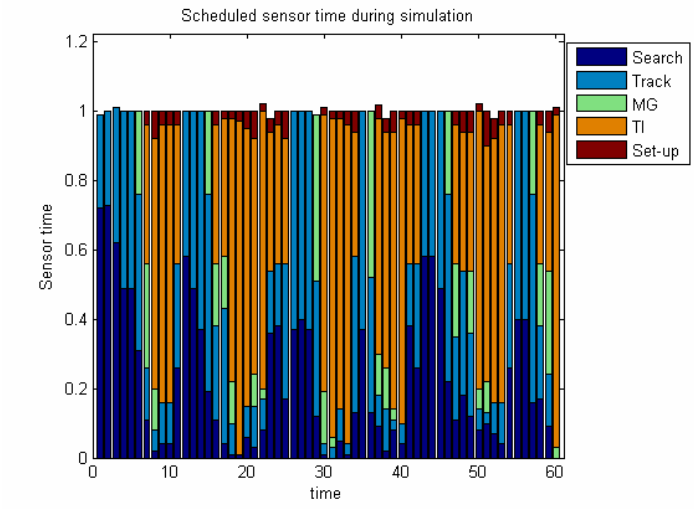

Figure 7.12 Sensor use with online GA scheduling

An observation that holds for all schedulers, and that was also seen in figures 7.7 – 7.9, is the number of executed (dropped) MG tasks. Before schedulers can be implemented in real life military systems this should be improved. The execution of a TI tasks becomes unnecessary if the midcourse guidance tasks were dropped for the particular weapon. Solution to this problem is adding (or increasing) the weighing factor for the different types of sensor functions.

7.5 Comparison

The results of the in total four schedulers can also be expressed using the evaluation parameters from section 5.6. The values for these measures are given in table 7.2.

Looking at the results of the utility, the choice would be to use the scheduler based on Lyapunov synthesis. This scheduler shows a high average utility. The extreme points in time also show that it performs at least as good as the FIFO scheduler when the demand is higher then the available budget. An additional advantage in the military domain is the system predictability: for equal demand, the resulting schedule is the same.

A scheduler that is perhaps faster due to fewer computations than the fuzzy Lyapunov based scheduler is the GNN. The simulation however showed some downsides of this approach. Table 7.2 shows that the scheduler drops more TI tasks than search tasks. In practise this means that some of the guided missiles will miss a target. Most likely this effect is due to the NN since the online GA doesn't have this side effect.

The online GA shows promising results. The utility is almost equal to that of the Lyapunov scheduler and it drops less TI tasks. In practise this means that more missiles will hit a target, thus reducing the risk to the mission.

Another advantage of the online GA is that the minimum utility in time is higher when compared to the Lyapunov scheduler. Again this means that it performs slightly better in complex scheduling situations. This is probably due to the lower percentage of dropped TI tasks. Also the effect of the parameter for set-up times can be seen in the low amount of time used for set-up, figure 7.12.

Table 7.2 Results of FIFO scheduler and the three developed schedulers

Evaluation parameter	FIFO	Lyapunov	GNN	Online GA
$U_{overall}$	*0.5938*	*0.6805*	*0.5730*	*0.6720*
$U_{\max} - U_{\min}$	*0.6437*	*0.6108*	*0.6732*	*0.5236*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ for search	*0.711*	*0.386*	*0.810*	*0.515*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ for track	*0.615*	*0.623*	*0.577*	*0.587*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ for MG	*0.507*	*0.604*	*0.524*	*0.510*
$\frac{\#\ executed\ \ tasks}{\#\ requested\ \ tasks}$ for TI	*0.502*	*0.866*	*0.534*	*0.891*

A final remark for all schedulers is the high percentage of dropped MG tasks compared to track tasks. Since the first are of more importance the equations could be adapted somewhat to ensure that MG tasks are dropped less. Furthermore, all three schedulers outperform the FIFO scheduler based on the percentage of dropped TI and MG tasks.

7.6 Improvements through hybridisation

To compare the hybrid scheduler with the other the parameters for the GA should be kept the same during several runs. Different tests have been done on the simulation with different values for the maximum number of generations and the population size. Assumed was that the hybrid form has a performance guarantee that at any given time its results are at least as good as of the composing schedulers. The hybrid scheduler is constructed using the fuzzy Lyapunov and FIFO scheduler in the online GA scheduler. Results can be found in appendix C and in figures 7.13 and 7.14. In the figures the results are shown for the scheduler with N=50 and maximum number of generations of 5.

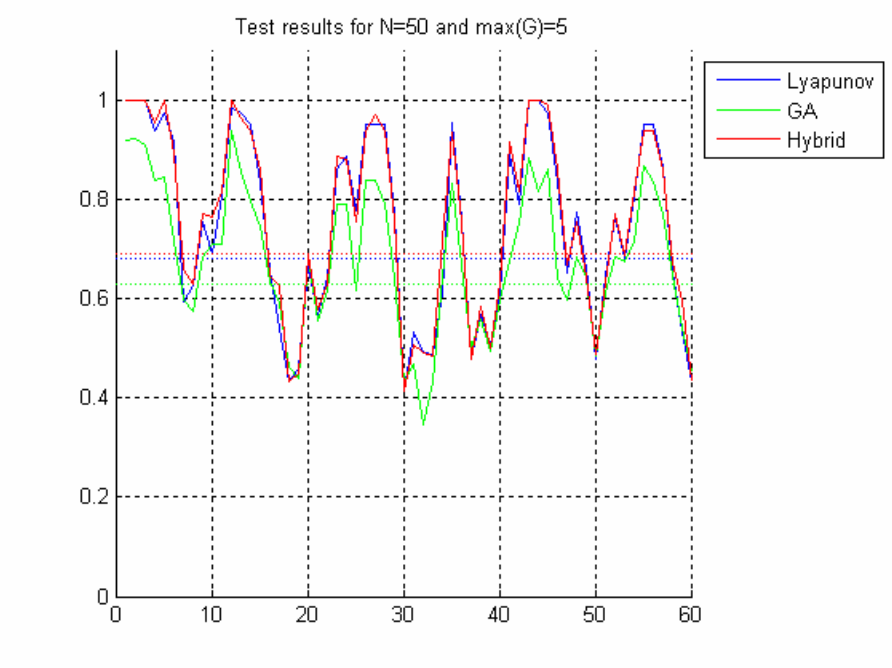

Figure 7.13 Utilities of the Lyapunov, the online GA and the hybrid schedulers

Figure 7.13 shows that the hybrid scheduler mostly follows the fuzzy Lyapunov based scheduler since the online GA doesn't have good performance due to the small numbers of generations and small initial population. This is exactly the desired behaviour: the hybrid GA has fast convergence and a guaranteed performance. Besides these factors the utility also shows that the hybridisation slightly improves the performance. At some points it outperforms the three schedulers that were used to make the hybrid scheduler. The reason for using a small population and few generations is illustrated by table 7.3 which shows the overall utility increase compared to the fuzzy Lyapunov scheduler, more detailed information can be seen in appendix C.

Table 7.3 Utility of hybrid scheduler minus utility of online GA scheduler

N	G		
	50	**25**	**5**
50	*0.0443*	*0.0453*	*0.0617*
60	*0.0408*	*0.0397*	*0.0545*
80	*0.0375*	*0.0485*	*0.0498*
100	*0.0333*	*0.0349*	*0.0448*

Figure 7.14 shows the performance gain by using a hybrid scheduler. The figure also illustrates that the hybrid scheduler uses the Lyapunov scheduler for performance guarantees and that it uses the evolving performance increase of the GA to improve where possible.

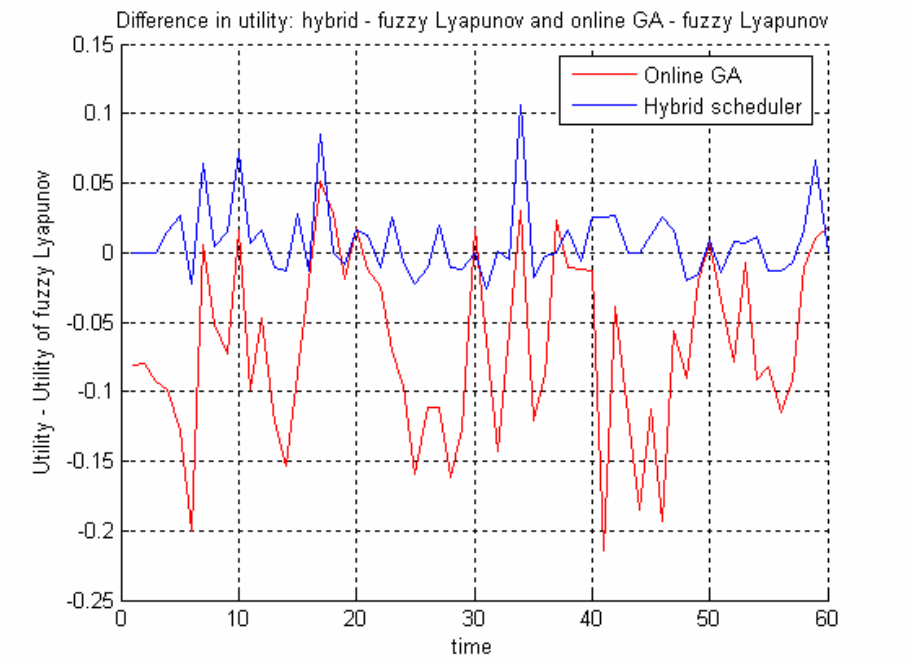

Figure 7.14 Comparing the online GA scheduler and the hybrid scheduler with the fuzzy Lyapunov based scheduler, N = 50 and the maximum number of generations is 5

Looking at the goals of the hybridisation it would seem that the hybrid scheduler can not be outperformed by any of the components. Figure 7.14 however shows that the fuzzy Lyapunov based scheduler has better performance at some points. The reason for this lies in the use of fuzzy Lyapunov in the hybrid scheduler.

Lyapunov gives a scheduling advice during runtime. When the hybrid scheduler finds a solution that increases performance, the generated advice by the fuzzy Lyapunov scheduler becomes invalid since it is then based on different buffer(s). Lyapunov has to be executed again to review its scheduling advice. In doing so, the hybrid Lyapunov component gives a different result than the stand alone Lyapunov scheduler. The overall performance however does improve so the gain in performance will most likely outweigh this negative effect, figure 7.14 also shows this.

A final remark on the hybrid scheduler is about time complexity. The stand alone schedulers like fuzzy Lyapunov and heuristics are fast. The online GA is only fast when using enough parallel processing. Hybridisation means that all schedulers should give their scheduling advice, which are converted to chromosomes after which a GA can be executed. Using heuristics and fuzzy Lyapunov decreases the needed time for generations within the online GA. The needed time to calculate all of theses advices from the different components before the GA starts however is time consuming. Temporal demands given by the real time character of sensor scheduling therefore means that this hybrid scheduler probably requires even more parallel processing power than the online GA by itself.

8 Conclusions & Recommendations

This thesis introduced three new scheduling methodologies and a hybrid scheduler for task scheduling in sensor networks. All of these schedulers are based on a novel view of the C2 process. The conclusions based on the hypotheses, as stated in section 4.8, are given in the first section of this chapter. The second section gives recommendations for further research.

8.1 Conclusions

This thesis identified and tested four promising techniques to solve the scheduling problem in sensor networks. All of these schedulers are based on an integration of the sensor allocation problem and task scheduling problem. After testing the four schedulers some conclusions can be drawn based on the hypotheses from section 4.8.

The Lyapunov scheduler proved to be a fast scheduling algorithm that can be implemented relatively easily. The results showed that it can outperform the often used first-in-first-out heuristic. This scheduler can be expanded to accommodate a multi sensor system. An advantage of this scheduler over the others is that it is predictable.

Genetic algorithms are a very strong optimisation tool. With sufficient computation power, this optimisation strength can be used directly in an online scheduler. This results in a high performance scheduler that can be made to be at least as good as any scheduler available today due to the copying without mutation of the fittest chromosomes. The online setting however should be developed and tested further to improve the results. An important factor in this scheduling method is that the results of the scheduler are very dependant of the number of generations that can be performed. Using parallel processing in this application can probably provide the necessary computation power required for the required number of generations.

The disappointing result of NN scheduling is most likely caused by the complexity of the sensor scheduling problem. Based on the training data no generalisation can be made causing the network to give scheduling results that are worse than the simple FIFO heuristic. A possible solution could be the use of more sophisticated training algorithms that enable the network to generalise better. Since the online implementation seems to work well this however might be unnecessary.

Hybridisation of different schedulers with the online GA shows a promising increase in performance. Real-time demands are met by using parallel processing.

8.2 Recommendations

Further work needs to be done in testing the schedulers in simulation environments. This would show their scheduling efficiency with respect to more realistic task sets and in real time. Another important test is to find the saturation points of the schedulers. This boundary indicates the possibility to track a certain number of objects while maintaining some search tasks as well.

More research could be done in optimising the chromosome mapping in the GA and in fine-tuning its parameters. For the fuzzy Lyapunov based scheduler further research could be done on the calculation of the buffer weight.

Work has to be done in developing better hybrid schedulers using the online GA setting. Using more heuristics and or more sophisticated heuristics might show an increase in performance while decreasing the needed time for generations in the GA.

References

[1] Alifantis T. and Robinson S., 'Using simulation and neural networks to develop a scheduler advisor', *Proceedings of the 2001 Winter Simulation Conference*

[2] Barbato A., Giustiniani P., 'An improved scheduling algorithm for a naval phased array radar', *Alenia Defence Systems, Italy*

[3] Bolderheij F. and Genderen, van P., 'Mission driven sensor management', *7th International Conference on Information Fusion, Stockholm, Sweden, 2004*

[4] Burbank A., 'Extracting beauty from chaos', 1999, http://plus.maths.org/issue9/features/lyapunov/

[5] Chen Z.L. and Powell W.B., 'Exact algorithms for scheduling multiple families of jobs on parallel machines', *Naval Research Logistics, 50:7, p823-840, 2003*

[6] Dauzére-Pérès S. and Sevaux M., 'Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine', *Navel Research Logistics, 50:3, p273-288, 2003*

[7] Duron C. and Proth J.M., 'Multifunction radar: Task scheduling', *Journal of Mathematical Modeling and Algorithms 1: 105–116, 2002*

[8] Huizing A.G. and Bloemen A.A.F., 'An efficient scheduling algorithm for a multifunction radar', *TNO-FEL, 2003*

[9] Hwang S.I. and Cheng S.T., 'Combinatorial optimization in real-time scheduling: Theory and algorithms', *Journal of Combinatorial Optimization, 5, 345–375, 2001*

[10] Jong, de J.L., 'Different representations for the traveling salesman problem using genetic algorithms', *Vrije Universiteit Amsterdam,*

[11] Lin K.Y., 'Decentralized admission control of a queueing system: A game-theoretic model', *Naval Research Logistics, 50:7, p702-718, 2003*

[12] Margialot M. and Langholz G., 'Design and analysis of fuzzy schedulers using fuzzy Lyapunov synthesis', *Engineering Applications of Artificial Intelligence, 14:2, p183-188, 2001*

[13] Mertens B., 'Reasoning with uncertainty in the situational awareness of air targets', *Masters thesis, Delft University of Technology 2004*

[14] Michalewicz Z., *Genetic Algorithms + data strucures = evolution programs,* Springer-Verlag, Berlin, Heidelberg, New York, 1996

[15] Negnevitsky M., '*Artificial intelligence: A guide to intelligent systems'*, Person Educational Limited, 2002

[16] Perkins J.R. and Kumar P.R., 'Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems', *IEEE Transactions on Automatic Control 34:2, p139-148, 1989*

[17] Pinto J.M. and Grossmann I.E., 'A logic-based approach to scheduling problem with resource constrants', *Computers and chemical engineering 21:8 p801-818, 1997*

[18] Policella N., Smith S.F., Cesta A.and Oddi A., 'Steps towards computing flexible schedules',

[19] Reeves C.R., *Modern heuristic techniques for combinatorial problems*, McGraw-Hill, 1995

[20] Schild K. and Wurtz J., 'Scheduling of time-triggered real-time systems', *Constraints, 5, p335-357, 2000*

[21] Thaens R. and Genderen, van P., 'Sensor scheduling using intelligent agents', *7th International Conference on Information Fusion, Stockholm, Sweden, 2004*

[22] Thaens R. and Genderen, van P., 'Task scheduling as strategic game', *2004, Unpublished*

[23] Thilakawardana S. and Tafazolli R., 'Use of genetic algorithms in efficient scheduling for multi service classes', 2004, http://research.ac.upc.es/EW2004/papers/15.pdf

[24] Weisstein E.W. et al. 'Lyapunov function' From *MathWorld* - A Wolfram Web Resource, http://mathworld.wolfram.com/LyapunovFunction.html

A Theoretical Background

Some background information is given in this appendix about the Lyapunov function, fuzzy logic, neural networks and genetic algorithms.

A.1 Lyapunov function

A Lyapunov function (as stated in [23]) is a scalar function $V(y)$ defined on a region D that is continuous, positive definite $V(y) > 0$ (for all $y \neq 0$) and has continuous first-order partial derivatives at every point of D. The existence of a Lyapunov function for which $V' \leq 0$ on some region D containing the origin, guarantees the stability of the zero solution of $y' = f(y)$.

For example, given the system:

y' = z;
z' = -y-2z.

If the Lyapunov function of the system is given by $V(y,z) = \frac{1}{2}\left(y^2 + z^2\right)$, the derivative is $V' = yz + z(-y - 2z) = -2z^2$. For each region of the scalars y and z this derivative is zero or negative meaning the system is stable.

Introducing fuzzy logic to a Lyapunov synthesis means that computing with words is possible. An example of such computing for solving scheduling problems is given in [12]. This paper also proves the stability of a scheduler based on fuzzy Lyapunov synthesis. Using this synthesis based on words means that all actions of the scheduler are based on rules that ensure the function derivative to be zero or negative

A.2 Fuzzy logic

Humans reason with vague concepts such as perhaps, a little, some etc., every day and are capable of dealing with it relatively easy in contrast to computers. Trying to make a system reason like humans a definition has to be given to these vague concepts. In 'normal' systems the question of something being 'close' is either *yes* (1) or *no* (0). This type of data representation is called crisp. The use of fuzzy sets however allows the computer to calculate a measure for being nearby, distant or far away. The basis for this ability lies in the membership functions. These functions give a certainty factor[1] of belonging to a state (such as nearby or distant) for a given interval of sensor data. By combining several fuzzy sets for control loops the resulting fuzzy logic system is very robust, i.e. it works well in noisy environments. In figure A.1 membership functions are drawn for the example of distance.

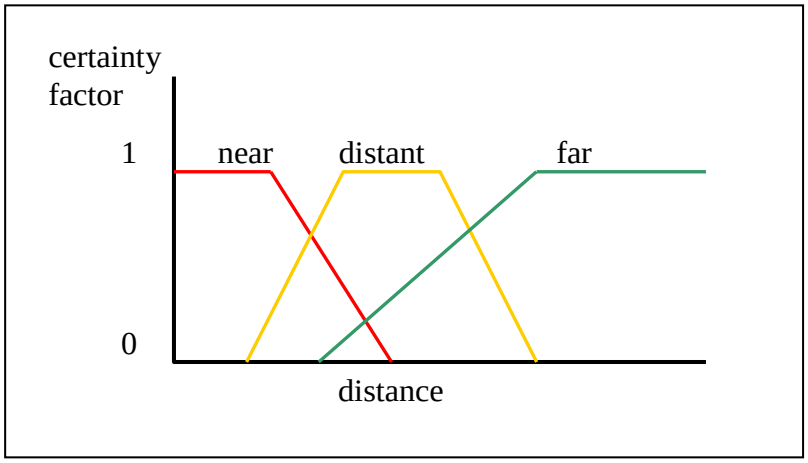

Figure A.1 Example membership function for distance

[1] Certainty factor (cf) Number between 0 and 1 stating the degree of truth in a fact. Different from probabilities because the sum of cfs need not be equal to 1.

A.3 Neural networks

The first research that was done in what is now called AI, was in neural networks (NN). The theory was based on physiology (function of neurons in the brain), propositional logic and Turings theory on computation. The model consists of artificial neurons that can be turned on or off based on the state of their neighbouring neurons. By modifying the strength of connections between neurons a NN can learn to mimic an operation when trained with appropriate data. In recent literature models based on NNs are called connectionist systems.

When trained a NN can generalize data and perform several tasks like pattern recognition or controlling a process. A drawback is that the outcome of the NN given a certain input is often unpredictable, the transparency is completely lost. The positive characteristic of a NN is that it works fast when used in an on-line process. All the work collecting training data and updating the neurons is done prior to run-time. By using on-line training strategies the NN can be made adaptive. Recent data is used to keep re-training the NN which causes the NN to adjust to changes in the environment.

A.4 Genetic Algorithms

Genetic algorithms (GA) are based on the process of natural selection and are used for optimisation problems. First a mapping has to be found to make so-called chromosomes out of the possible solutions. Secondly a fitness function has to be made in order to evaluate the chromosomes. After that the process of evolution can begin.

An initial population is made consisting of several randomly chosen chromosomes. From these chromosomes the fittest are chosen to create offspring. By using the fitness function on the initial population and their offspring the strongest chromosomes remain in the population, others will become extinct, because the size of the population can’t change over the course of generations. After a predefined number of generations the fittest chromosome is chosen as the optimal solution to the problem. For more reading on GAs the reader is referred to [14].

The general problem when applying GAs in optimization problems is speed. Generating all the chromosomes with cross-over schemes and comparing finesses is takes a lot of computational power. The power of GAs is that they are very useful in finding optimal solutions in very large search spaces.

B Table with simulation objects

Here the simulation objects are represented in time with their priorities. The objects are divided over 17 columns, the first nine in part 1 of the table and the other in part two. In section 7.1 the simulation is discussed and there 18 missiles are mentioned. These are placed in three columns since a missile appears only for a short period of time.

Used abbreviations in the table:

HS : Horizon Search;
LVS : Limited Volume Search;
MG : Midcourse Guidance;
TI : Terminal Illumination;
Tr : Track;
M1 : First record for a missile;
M2 : Second record for a missile;
M3 : Third record for a missile.

B.1 Part 1

Task Duration (s)	0.12	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.04
Update Rate (Hz)	1	2	2	2	2	5	5	8	10
Task type	**HS**	**LVS**	**LVS**	**LVS**	**LVS**	**Tr (plane)**	**Tr (M1)**	**MG**	**TI**
Time(s)	**Priority**								
1	0.5431	0.3	0.4	0.6	0.2	0.6	0	0	0
2	0.5431	0.298	0.402	0.62	0.202	0.602	0	0	0
3	0.5431	0.296	0.404	0.64	0.204	0.604	0.99	0	0
4	0.5431	0.294	0.406	0.66	0.206	0.606	0.99	0	0
5	0.5431	0.292	0.408	0.68	0.208	0.608	0.99	0	0
6	0.5431	0.29	0.41	0.7	0.21	0.61	0.99	0	0
7	0.5431	0.288	0.412	0.72	0.212	0.612	0.99	1	0
8	0.5431	0.286	0.414	0.74	0.214	0.614	0	1	1
9	0.5431	0.284	0.416	0.76	0.216	0.616	0	0	1

10	0.5431	0.282	0.418	0.78	0.218	0.618	0	0	1
11	0.5431	0.28	0.42	0.8	0.22	0.62	0	0	1
12	0.5431	0.278	0.422	0.82	0.222	0.622	0	0	0
13	0.5431	0.276	0.424	0.84	0.224	0.624	0.99	0	0
14	0.5431	0.274	0.426	0.86	0.226	0.626	0.99	0	0
15	0.5431	0.272	0.428	0.858	0.228	0.628	0.99	0	0
16	0.5431	0.27	0.43	0.856	0.23	0.63	0.99	0	0
17	0.5431	0.268	0.432	0.854	0.232	0.632	0.99	1	0
18	0.5431	0.266	0.434	0.852	0.234	0.634	0	1	1
19	0.5431	0.264	0.436	0.85	0.236	0.636	0	0	1
20	0.5431	0.262	0.438	0.848	0.238	0.638	0	0	1
21	0.5431	0.26	0.44	0.846	0.24	0.64	0	0	1
22	0.5431	0.258	0.442	0.844	0.242	0.642	0	0	0
23	0.5431	0.256	0.444	0.842	0.244	0.644	0	0	0
24	0.5431	0.254	0.446	0.84	0.246	0.646	0	0	0
25	0.5431	0.252	0.448	0.838	0.248	0.648	0.99	0	0
26	0.5431	0.25	0.45	0.836	0.25	0.65	0.99	0	0
27	0.5431	0.248	0.452	0.834	0.252	0.652	0.99	0	0
28	0.5431	0.246	0.454	0.832	0.254	0.654	0.99	0	0
29	0.5431	0.244	0.456	0.83	0.256	0.656	0.99	1	0
30	0.5431	0.242	0.458	0.828	0.258	0.658	0	1	1
31	0.5431	0.24	0.46	0.826	0.26	0.66	0	0	1
32	0.5431	0.238	0.462	0.824	0.262	0.662	0.99	0	1
33	0.5431	0.236	0.464	0.822	0.264	0.664	0.99	0	1
34	0.5431	0.234	0.466	0.82	0.266	0.6	0.99	0	0
35	0.5431	0.232	0.468	0.818	0.268	0.59	0.99	0	0
36	0.5431	0.23	0.47	0.816	0.27	0.58	0.99	1	0
37	0.5431	0.228	0.472	0.814	0.272	0.57	0	1	1
38	0.5431	0.226	0.474	0.812	0.274	0.56	0	0	1
39	0.5431	0.224	0.476	0.81	0.276	0.55	0	0	1
40	0.5431	0.222	0.478	0.808	0.278	0.54	0	0	1
41	0.5431	0.22	0.48	0.806	0.28	0.53	0	0	0
42	0.5431	0.218	0.482	0.804	0.282	0.52	0	0	0
43	0.5431	0.216	0.484	0.802	0.284	0.51	0	0	0
44	0.5431	0.214	0.486	0.8	0.286	0.5	0	0	0
45	0.5431	0.212	0.488	0.798	0.288	0.49	0.99	0	0
46	0.5431	0.21	0.49	0.796	0.29	0.48	0.99	0	0
47	0.5431	0.208	0.492	0.794	0.292	0.47	0.99	0	0
48	0.5431	0.206	0.494	0.792	0.294	0.46	0.99	0	0
49	0.5431	0.204	0.496	0.79	0.296	0.45	0.99	1	0
50	0.5431	0.202	0.498	0.788	0.298	0.44	0	1	1
51	0.5431	0.2	0.5	0.786	0.3	0.43	0	0	1
52	0.5431	0.198	0.502	0.784	0.302	0.42	0	0	1
53	0.5431	0.196	0.504	0.782	0.304	0.41	0	0	1
54	0.5431	0.194	0.506	0.78	0.306	0.4	0	0	0
55	0.5431	0.192	0.508	0.778	0.308	0.39	0.99	0	0
56	0.5431	0.19	0.51	0.776	0.31	0.38	0.99	0	0
57	0.5431	0.188	0.512	0.774	0.312	0.37	0.99	0	0

58	0.5431	0.186	0.514	0.772	0.314	0.36	0.99	0	0
59	0.5431	0.184	0.516	0.77	0.316	0.35	0.99	1	0
60	0.5431	0.182	0.518	0.768	0.318	0.34	0	1	1

B.2 Part 2

Task Duration (s)	0.03	0.03	0.04	0.03	0.03	0.04	0.03	0.03
Update Rate (Hz)	5	8	10	5	8	10	5	5
Task type	**Tr (M2)**	**MG**	**TI**	**Tr(M3)**	**MG**	**TI**	**Tr(Hostile)**	**Tr(Friendly)**
Time(s)	**Priority**							
1	0	0	0	0	0	0	0.7	0.2
2	0	0	0	0	0	0	0.702	0.2
3	0	0	0	0	0	0	0.704	0.2
4	0.99	0	0	0	0	0	0.706	0.2
5	0.99	0	0	0	0	0	0.708	0.2
6	0.99	1	0	0	0	0	0.71	0.2
7	0	1	1	0	0	0	0.712	0.2
8	0	0	1	0	0	0	0.714	0.2
9	0	0	1	0	0	0	0.716	0.2
10	0	0	1	0	0	0	0.718	0.2
11	0	0	0	0.99	0	0	0.72	0.2
12	0	0	0	0.99	0	0	0.722	0.2
13	0	0	0	0.99	0	0	0.724	0.2
14	0.99	0	0	0.99	0	0	0.726	0.2
15	0.99	0	0	0.99	1	0	0.728	0.2
16	0.99	0	0	0	1	1	0.73	0.2
17	0.99	0	0	0.99	0	1	0.732	0.2
18	0.99	1	0	0.99	0	1	0.734	0.2
19	0	1	1	0.99	0	1	0.736	0.2
20	0	0	1	0.99	0	0	0.738	0.2
21	0	0	1	0.99	1	0	0.74	0.2
22	0	0	1	0	1	1	0.742	0.2
23	0	0	0	0	0	1	0.744	0.2
24	0	0	0	0	0	1	0.746	0.2
25	0	0	0	0.99	0	1	0.748	0.2
26	0.99	0	0	0.99	0	0	0.75	0.2
27	0.99	0	0	0.99	0	0	0.752	0.2
28	0.99	0	0	0.99	0	0	0.754	0.2
29	0.99	0	0	0.99	1	0	0.756	0.2
30	0.99	1	0	0	1	1	0.758	0.2
31	0	1	1	0	0	1	0.76	0.2
32	0	0	1	0.99	0	1	0.762	0.2
33	0	0	1	0.99	0	1	0.764	0.2
34	0.99	0	1	0.99	0	0	0.766	0.2
35	0.99	0	0	0.99	0	0	0.768	0.2

36	0.99	0	0	0.99	1	0	0.77	0.2
37	0.99	0	0	0	1	1	0.772	0.2
38	0.99	1	0	0	0	1	0.774	0.2
39	0	1	1	0	0	1	0.776	0.2
40	0	0	1	0	0	1	0.778	0.2
41	0	0	1	0	0	0	0.78	0.2
42	0.99	0	1	0	0	0	0.782	0.2
43	0.99	0	0	0	0	0	0.784	0.2
44	0.99	0	0	0	0	0	0.786	0.2
45	0.99	0	0	0	0	0	0.788	0.2
46	0.99	1	0	0.99	0	0	0.79	0.2
47	0	1	1	0.99	0	0	0.792	0.2
48	0	0	1	0.99	0	0	0.794	0.2
49	0	0	1	0.99	0	0	0.796	0.2
50	0	0	1	0.99	1	0	0.798	0.2
51	0	0	0	0	1	1	0.8	0.2
52	0	0	0	0	0	1	0.802	0.2
53	0.99	0	0	0	0	1	0.804	0.2
54	0.99	0	0	0	0	1	0.806	0.2
55	0.99	0	0	0.99	0	0	0.808	0.2
56	0.99	0	0	0.99	0	0	0.81	0.2
57	0.99	1	0	0.99	0	0	0.812	0.2
58	0	1	1	0.99	0	0	0.814	0.2
59	0	0	1	0.99	1	0	0.816	0.2
60	0	0	1	0	1	1	0.818	0.2

C Results of hybrid scheduler

	Lyapunov	N = 100					
		max(G) = 50		max(G) = 25		max(G) = 5	
		online GA	Hybrid	online GA	Hybrid	online GA	Hybrid
Utility	*0.6815*	*0.6564*	*0.6897*	*0.653*	*0.6879*	*0.6451*	*0.6899*
Umax - Umin	*0.5857*	*0.5176*	*0.5857*	*0.4873*	*0.5857*	*0.5218*	*0.5857*
executed search	*0.3864*	*0.5958*	*0.4606*	*0.6064*	*0.4663*	*0.6314*	*0.4508*
executed track	*0.6041*	*0.5367*	*0.6051*	*0.5439*	*0.6*	*0.5408*	*0.6112*
executed MG	*0.5868*	*0.4861*	*0.5903*	*0.5069*	*0.5868*	*0.5035*	*0.5868*
executed TI	*0.8938*	*0.9015*	*0.8785*	*0.8754*	*0.8815*	*0.8508*	*0.88*

	Lyapunov	N = 80					
		max(G) = 50		max(G) = 25		max(G) = 5	
		online GA	Hybrid	online GA	Hybrid	online GA	Hybrid
Utility	*0.6815*	*0.6496*	*0.6871*	*0.6408*	*0.6893*	*0.6402*	*0.69*
Umax - Umin	*0.5857*	*0.5864*	*0.5857*	*0.723*	*0.6029*	*0.5457*	*0.5857*
executed search	*0.3864*	*0.6189*	*0.4655*	*0.6299*	*0.4557*	*0.6379*	*0.4511*
executed track	*0.6041*	*0.5337*	*0.6*	*0.5265*	*0.6143*	*0.5337*	*0.6153*
executed MG	*0.5868*	*0.5069*	*0.5833*	*0.4687*	*0.5868*	*0.5*	*0.6007*
executed TI	*0.8938*	*0.8723*	*0.88*	*0.8831*	*0.8738*	*0.8523*	*0.8708*

	Lyapunov	N = 60					
		max(G) = 50		max(G) = 25		max(G) = 5	
		online GA	Hybrid	online GA	Hybrid	online GA	Hybrid
Utility	*0.6815*	*0.6489*	*0.6897*	*0.6458*	*0.6855*	*0.635*	*0.6895*
Umax - Umin	*0.5857*	*0.514*	*0.5857*	*0.5831*	*0.5857*	*0.6499*	*0.5857*
executed search	*0.3864*	*0.6182*	*0.4568*	*0.6273*	*0.4712*	*0.6432*	*0.4587*
executed track	*0.6041*	*0.5337*	*0.6092*	*0.5439*	*0.601*	*0.5296*	*0.6163*
executed MG	*0.5868*	*0.4861*	*0.5903*	*0.4896*	*0.5764*	*0.4757*	*0.5903*
executed TI	*0.8938*	*0.8785*	*0.8769*	*0.8585*	*0.8738*	*0.8523*	*0.8677*

	Lyapunov	N = 50					
		max(G) = 50		max(G) = 25		max(G) = 5	
		online GA	Hybrid	online GA	Hybrid	online GA	Hybrid
Utility	*0.6416*	*0.6416*	*0.6859*	*0.6432*	*0.6885*	*0.6286*	*0.6903*
Umax - Umin	*0.5742*	*0.5742*	*0.5857*	*0.5681*	*0.6029*	*0.5898*	*0.5857*
executed search	*0.6295*	*0.6295*	*0.4576*	*0.6333*	*0.4591*	*0.6561*	*0.4424*
executed track	*0.5388*	*0.5388*	*0.5959*	*0.5408*	*0.6041*	*0.5327*	*0.6224*
executed MG	*0.4687*	*0.4687*	*0.5729*	*0.4896*	*0.5799*	*0.4826*	*0.5972*
executed TI	*0.8662*	*0.8662*	*0.8923*	*0.8538*	*0.8831*	*0.8323*	*0.8708*

Intelligent task scheduling in sensor networks

Wilbert van Norden[*,***] **Jeroen de Jong**[**] **Fok Bolderheij**[***] **Léon Rothkrantz**[*]

[*]Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Man-Machine Interaction group
P.O. box 5031, 2600 GA Delft, The Netherlands

[**]Thales Research & Technology
Delft Cooperation on Intelligent Systems
P.O. box 90, 2600 AB Delft, The Netherlands

[***]Royal Netherlands Naval College
Combat Systems Department
P.O. box 10000, 1780 CA Den Helder, The Netherlands

Abstract: *Ever more complex sensors have become available to create and maintain situational awareness during missions. Choosing the most suited sensor for the execution of a sensor function is based on sensor capabilities and function attributes. When these characteristics change rapidly, sensor allocation for sensor functions will shift. To increase performance of the entire sensor network, the total set of sensors should be scheduled in a single system. This paper puts forward and compares three new methods for scheduling prioritised tasks in sensor networks. The first is based on fuzzy Lyapunov synthesis. The other two use a genetic algorithm (GA) to optimise the set of schedules. The second scheduler uses this set to (re)train a neural network to be used online. The third approach is a novel online use of the GA. Tests showed that the novel online GA leads to a robust scheduling algorithm with high overall performance.*

Keywords: Sensor Management, Task Scheduling, Command & Control, Online Genetic Algorithms, fuzzy Lyapunov

1 Introduction

On the new Air Defence and Command Frigates of the Royal Netherlands Navy the APAR (Active Phased Array Radar) Multi Function Radar (MFR) is used for anti air and surface warfare. This system is able to perform multiple sensor functions like search, track and weapon guidance. Since more sensor functions can now be performed by a single sensor the underlying management structure has increased in complexity.

Looking for ways to improve MFR performance, it is obvious to look at the scheduling algorithm that decides which sensor task will be performed when (or which task should be dropped). This scheduling problem is known as the 'radar scheduling problem'. In light of the increasing interest in Network Centric Warfare (NCW), the properties of a scheduling algorithm for a single MFR should be extended to schedule sensor functions in a sensor network.

Currently, all tasks are generated by the combat management system and sent to the sensor chosen by the operator. Scheduling is done on the sensor level, using operator input to define the priority of sensor tasks. An undesired side effect of this way of scheduling is that sensor tasks might be dropped while they could have been executed by a less appropriate sensor. Making schedules for the entire set of sensors instead of optimising for each sensor separately seems to be a more fitting approach.

To do so, we propose three methodologies. The first is based on the scheduling algorithm by Margialot and Langholz [7]. They use fuzzy Lyapunov synthesis for solving a single machine job shop scheduling problem.

The other two algorithms are both based on a genetic algorithm (GA). We developed a new chromosome mapping and a fitness function enabling the use of genetic operators. The second algorithm uses the GA offline to find optimal solutions of past problems. These optimal solutions are used to (re)train a neural network (NN) that is used online for the actual scheduling. Such an approach results in an adaptive scheduler that combines the optimisation power of GAs and the speed of NNs.

The last scheduler is solely based on our GA. We developed a way to use the GA online without the setback of needing too much computation time.

Current schedulers and some developments in this field are discussed in section two. Section three of this paper describes a novel view on the command and control process used aboard navy ships. This view is expanded to the sensor scheduling level in section four. Using this approach, the scheduling algorithms can be discussed. Section five discusses the scheduler based on fuzzy Lyapunov synthesis.

Our genetic algorithm is discussed in section six. After this the NN approach and the online use are discussed in sections seven and eight. Some tests were done on all schedulers and are described in section nine. Finally, conclusions are drawn in section ten.

2 Related work

Much work has already been done in the field of radar (sensor) scheduling, especially since the introduction of the MFR. Some of these schedulers are based on a rotating MFR [2] and [4]. Scheduling tasks within a phased array MFR however is very different. Huizing and Bloemen proposed a scheduler based on queuing of dwell requests [5]. Our goal is to find a scheduling methodology that can be expanded to handle multiple sensors. For that reason this queuing strategy will not work directly.

Another new approach is to use a multi agent system for task scheduling, proposed by Thaens in [13]. Much work needs to be done on the negotiating scheme, rendering this approach not yet applicable for implementation in current sensor networks.

A promising scheduler is the OGUPSA algorithm, from McIntyre and Hintz, [8]. Most of the requirements for a good scheduler (like dealing with multiple sensors and real-time demands) are considered in their approach. A downside however is the sensor allocation: they look only at the ability of a sensor to perform a certain task, whereas the quality of the execution should also be considered.

To find a scheduler for the sensor network problem we therefore looked at another scheduling domain, namely the job shop scheduling problem. Algorithms used in that domain can be adapted to fit the sensor domain. For speed and optimality reasons we chose to implement a scheduler based on fuzzy Lyapunov synthesis (Margialot and Langholz [7]) and NN scheduling (Alifantis and Robinson [1]).

Since GAs are a strong optimisation tool in large search spaces they are chosen and adapted to generate the training data for the NN, an example of GAs in scheduling is e.g. Thilakawardana and Tafazolli [14].

3 Command & Control

Generally the command and control (C2) processes are modelled based on the well known OODA-loop. In [3], Bolderheij proposes to prioritise sensor tasks using risk estimation on threat objects. Both search and track tasks can then be scheduled based on the same principles. Introducing this concept in C2 leads to parallel processes instead of a loop, illustrated in Figure 1.

This new approach shows that sensor scheduling is only dependent on an ‘object store’ containing the available information on all detected and expected objects. All objects have a priority based on the risk they pose to the mission.

Figure 1 shows that the objects are the central point in the C2 process. All processes use some of the information held by the objects’ attributes update parts of it. The classification process for instance, uses the characteristics of an object to calculate the probability of this object belonging to certain

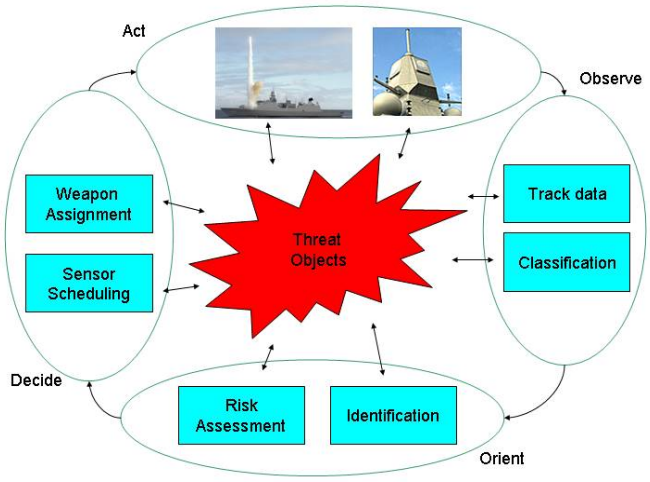

Figure 1 Command and control process aboard navy ships

class, see Mertens [9]. If the sensor manager notices that an object has a high uncertainty in classification, it can request sensor functions to update the relevant information about the object in order to help the classification process. This example shows that even though the different processes are decoupled, interdependency is maintained through the ‘object store’.

An important attribute of an object is the risk it poses with respect to the completion of a mission. Requests for sensor functions get a priority equal to the risk posed by the object the task refers to.

4 Sensor management

Since the C2 processes are considered as a set of parallel systems, we can look at the sensor scheduling in detail without looking at other processes. This leads to the resulting architecture of sensor network scheduling shown in Figure 2.

Optimising the different processes in Figure 2 one at a time, might not lead to the best picture compilation of the environment or to minimisation of risk during the mission. To reach an optimum the entire chain of resource allocation, task scheduling and dwell scheduling needs to be optimised. Dwell scheduling however is difficult to include since it is very sensor dependent and because the manufacturer of the sensor develops this part. We therefore only consider the combined problem of resource allocation and task scheduling. Since tasks are generated from the object store they are immediately available to the scheduler. The complexity of the ‘release date’ is therefore eliminated. The task attributes of importance to the scheduler are:

i) type of sensor function;
ii) priority;
iii) due date;
iv) duration;
v) position of the object to determine if it is in range for the sensors.

In order to develop an optimal scheduler we first look at the single MFR scheduling problem. If a successful scheduler for this type of sensor can be constructed we will extend this scheduler for a sensor network on a single ship. The ultimate goal, of course, is to schedule a distributed set of sensors, which is the case in NCW.

Scheduling a sensor network requires performance indicators to determine which sensor is best fitted for a sensor task at a given time. This is necessary for the single ship, multi sensor case as well as in NCW. By adding a range component the single ship problem can be expanded to a multi ship problem. Therefore the assumption can be made that this final expansion isn't very complex and that most problems will arise in expanding from single sensor to a sensor network.

5 Fuzzy Lyapunov

Figure 2 shows that the scheduling process for sensor(s) starts with a task request. This view on sensor scheduling can be compared with solving JSSPs. Margialot and Langholz, [7], introduced a scheduler based on fuzzy Lyapunov synthesis. This approach resembles a heuristic solution with the advantage of the stability offered by the Lyapunov synthesis.

The general idea here is to create different buffers for the different types of sensor functions. Whenever a sensor becomes available to execute a new task, the scheduler decides from which buffer the new task should come. To do so, it calculates the weight of the buffers (B_W) and then chooses the buffer with the highest weight. Since Margialot and Langholz assumed that all tasks have the same priority the formula used in [7] is unsuited for the domain of sensor scheduling. We therefore redefined the buffer weights as given in Eq. (1).

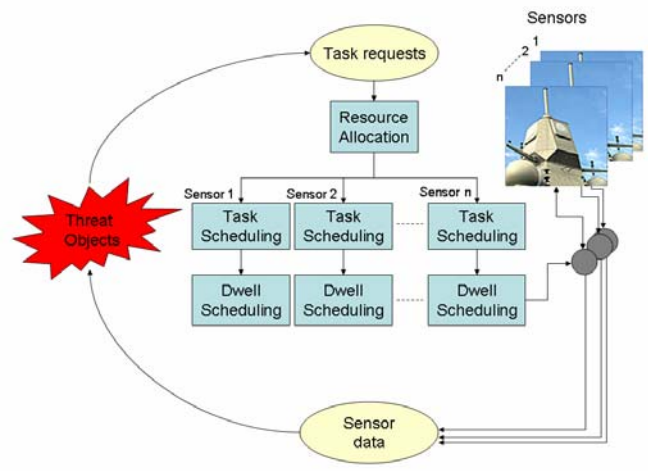

Figure 2 Architecture of control loop for a sensor network

$$B_W = k \cdot \sqrt{X} \cdot \sum_{\forall i} \rho_i P_i \left(\frac{(1-\alpha)\cdot \tau_i + \alpha}{\gamma \cdot \left(t_{dd,i} - t_{ct} - t_{td,i}\right)} \right) \qquad (1)$$

In Eq. (1) the following denotations are used:

k	:	Weight factor of the buffer;
X	:	Number of tasks in the buffer;
ρ_i	:	Range factor of object i, 1 if in range of the sensor else 0;
P_i	:	Priority of task i in the buffer, $0 \le P_i \le 1$;
τ_i	:	Relative processing time of task i in the buffer, $0 \le \tau_i \le 1$;
$t_{dd,i}$	:	Due date of task i in the buffer;
t_{ct}	:	Current time;
$t_{td,i}$	:	Task duration of task i in the buffer;
α	:	Weight factor to map relative processing time to a different domain[1], $[0,1] \rightarrow [\alpha,1]$, of course $0 \le \alpha \le 1$ holds;
γ	:	Weight factor for due date.

Extending the weight allocation from a single machine (single MFR) solution to multiple machines (sensor network) is relatively easy. The scheduler calculates the buffer weights whenever a sensor is available to receive a new task. By using the sensor specific values for k, τ and $t_{td,i}$ each sensor receives the task type for which it is most suited at that time.

After finding the right buffer, a task has to be selected from it. This mechanism is also straightforward: select the task that has the highest contribution to the buffer weight is selected. Since Lyapunov synthesis is based on stability analysis and we choose the task with highest contribution to this weight the number of tasks in the buffers is guaranteed to be limited.

6 Genetic Algorithm

Genetic algorithms are often used to find good solutions in search spaces that are too big to be searched exhaustively. Given the size of the search space[2] for scheduling in sensor networks GAs look promising. There are some issues when using a GA that need to be solved prior to applying GA for any problem.

[1] This makes it possible to make the priority more important than the required processing time.

[2] E.g.: 3 tasks on two sensors can be scheduled in 24 different ways, 4 tasks on two sensors in 120 ways

Two of these issues are inherent to using GAs: defining a fitness function and a chromosome mapping. The third is the speed of the algorithm. Although it doesn't take as much time as searching through the entire solution space, it is still computational complex, rendering it inapplicable for direct use in sensor scheduling due to the near-real-time requirements of sensor scheduling. Dealing with this last issue is discussed in sections six and seven.

6.1 Fitness function

Before we can use the GA for optimisation we have to define the fitness function C_F, Eq. (2). This function has to be maximised to find the optimal schedule. The downside of maximisation here is that the maximum is unknown, meaning that when we stop after a certain amount of generations we can't say how (sub) optimal the solution is.
The choice for maximisation was made since the goal is to execute as many tasks with high priority on the most suited sensor. A minimisation function could be made that should go to zero. However, it still couldn't be said how close to zero the most optimal schedule can get since reaching zero is not always possible.

$$C_F = \sum_{j=1}^{N} \sum_{i=1}^{n_j} \kappa_{j,i} \delta_{i,j} S_{j,i} P_i \left((1-\alpha) \cdot \tau_{i,j} + \alpha \right) \qquad (2)$$

In Eq. (2) the following denotations are used:

C_F	:	Chromosome fitness;
$\kappa_{j,i}$	:	Appropriateness of sensor j for task i, $0 \le \kappa_{j,i} \le 1$;
$\delta_{i,j}$	:	Feasibility of task i on sensor j, 1 if $t_{ct} + t_{td} < t_{dd}$ else 0;
$S_{j,i}$	:	Set-up factor in sensor j for task i, 1 if no set-up is needed else 0.8;
P_i	:	Priority of tasks i, $0 \le P_i \le 1$;
$\tau_{i,j}$	:	Relative processing time of task i on sensor j, $0 \le \tau_{i,j} \le 1$;
α	:	Weight factor to map relative processing time on a different domain, $[0,1] \rightarrow [\alpha,1]$, with $0 \le \alpha \le 1$;
n_j	:	Number of tasks for sensor j;
N	:	Number of sensors in the network.

6.2 Chromosome mapping

In order to be able to use genetic operators, the mapping between solution and chromosome should be well defined. Chromosomes should facilitate the use of operations like cross-over and mutation. After these operations the schedules embedded in the new chromosomes should still be valid. To obtain a good mapping we need to find a representation of the problem at hand. First we look at a representation for the single sensor problem. This mapping will then be extended to also map the multiple sensor case.

Assume that at a given time all task requests are placed in a single buffer with size *K* and the GA finds the optimal schedule for this buffer. A schedule can then be represented as a string of integers with values between 1 and *K*. In such a schedule each value can only appear once since each value in the schedule refers to a position in the buffer.

This mapping has a major advantage. Looking at the constraints placed on the representation of the schedule a good comparison can be made with the well known travelling salesman problem. Much research has already been conducted in solving this type of problem, so a mapping strategy already exists. The mapping we use is the swapping representation from De Jong [6].

This however solves only part of the mapping problem. In the multi sensor context we want to assign a sensor to certain tasks. This problem is solved by adding values to the schedule.

Consider ten tasks to be scheduled, $K = 10$, on three sensors. The schedule is mapped with swapping representation to chromosome *Chr*. For each sensor an element is added that says which part of *Chr* is to be executed by what sensor. The schedule, *S*, for the set of sensors is obtained by using the swapping representation denoted by $S = swapping(Chr)$. For multiple sensors, S_j denotes the schedule for sensor *j* with $S = \langle S_1 \oplus S_2 \oplus ... \oplus S_N \rangle$.

$$chromosome = \langle [4 \quad 3 \quad 3] \quad [Chr] \rangle$$
$$\begin{cases} S_1 = elements \quad 1 \rightarrow 4 \quad of \quad swapping(Chr) \\ S_2 = elements \quad 5 \rightarrow 7 \quad of \quad swapping(Chr) \\ S_3 = elements \quad 8 \rightarrow 10 \quad of \quad swapping(Chr) \end{cases}$$

This representation however isn't a good mapping since cross-over and mutation are still impossible since the added values must equal *K* when summed. Looking at the additional elements as weights rather than numbers solves this problem.

$$chromosome = \langle [2 \quad 7 \quad 4] \quad [Chr] \rangle$$
$$\begin{cases} S_1 = elements \quad 1 \rightarrow a \quad of \quad S \\ S_2 = elements \quad a+1 \rightarrow a+b \quad of \quad S \\ S_3 = elements \quad b+1 \rightarrow a+b+c \quad of \quad S \\ a+b+c \equiv K \end{cases}$$

$$a = floor\left(\frac{2 \cdot K}{2+7+4}\right) = floor\left(\frac{20}{13}\right) = 1$$

$$b = floor\left(\frac{7 \cdot K}{2+7+4}\right) = floor\left(\frac{70}{13}\right) = 5$$

$$c = floor\left(\frac{4 \cdot K}{2+7+4}\right) = floor\left(\frac{40}{13}\right) = 3$$

while $a+b+c < K$

add 1 *to smallest value*

end

$\rightarrow \quad a = 2 \quad b = 5 \quad c = 3$

This type of mapping enables cross-over and mutation. Since this mapping is highly dependent on the problem domain, the construction of new generations should be made to fit the problem.

Note that this is a simple mapping that works. Other mappings, e.g. the mapping used in vehicle routing problem (see [11]), might prove to result in a faster convergence to maximal chromosome fitness. We will not discuss other mappings since the goal here is only to give a conceptual proof of using GA in sensor scheduling.

6.3 Creating the new population

The new population is generated from the old population with the use of cross-over and mutation. The biggest part of the new population, 65%, is made by using cross-over on the chromosomes with highest fitness. On this part a mild form of mutation is applied. The next part is constructed by copying the fittest chromosomes from the old population into the new population. On the fittest chromosomes in this group no mutation is applied. On the others the elements for sensor allocation are mutated.

The final part of the new population is made by adding new randomly generated chromosomes. These random chromosomes build up 5% of the new population and are added to avoid local optima and to introduce new genetic material for cross-over in the next generation.

The values for these percentages were determined by tests and seemed to give good results. Before GAs can be used for sensor scheduling more tests should be done to optimise the construction of new populations. This however is also dependent on the optimal mapping solution.

7 Training a neural network

As mentioned in section five, the down side of a GA is the computation time needed to find optimal solutions. This problem can be solved by running the GA in the background and use its solutions to train a feed-forward NN, Figure 3.

To train a NN, we used the back propagation algorithm. Different network configurations can be used depending on user input. The number of in- and output nodes are of course defined by the size of the sensor network and the number of tasks in the buffer.

For the hidden layers the user can choose the number of nodes freely. The number of hidden layers can be chosen with a maximum of three. This choice was made since most practical systems never use more than three hidden layers, see [10]. The number of nodes and hidden layers can be chosen but the network structure is of course based on the general theories discussed in e.g. [12].

Figure 3 shows the general outline of the resulting scheduler. Incoming task request are placed in a buffer. This buffer is then scheduled by the NN. Over-training is prevented by using the GA offline to find optimal schedules. These solutions are used to retrain another NN that replaces the NN used online when it is fully trains. The result is a fast and adaptive scheduler.

8 Online use of GA

Using our GA for training NNs is not the only option to overcome the problem of time constraints. Online use of the GA was accomplished by using a buffering scheme similar to the one used in the Lyapunov based scheduler. Here only one buffer is used with a fixed size *K*. This buffer is filled with default tasks until task requests are received. These requests then replace default tasks in the buffer.

Whenever a sensor becomes available to execute a new task the scheduler looks at the current best solution given by the GA. It identifies the correct schedule for the sensor and only uses the first entry, meaning that only one task is sent to the sensor. Doing so means the buffer only changes slightly. Since the GA is constantly running, only few generations are needed to find a new optimal solution.

In this setting the GA does not have any stop criteria. We therefore can say nothing about the optimality of the schedule that is used. Although this might seem to be a large setback, it most likely is not due to the real-time demands on the system. In the military domain a sub-optimal solution in time is better than an optimal solution that is too late.

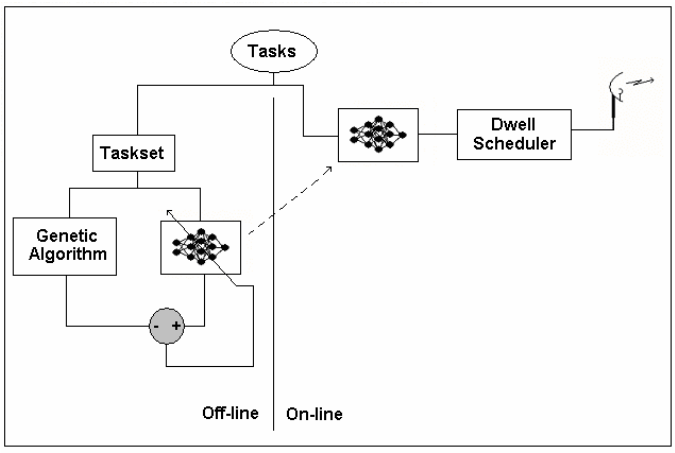

Figure 3 Architecture of the genetic-neural network

The solutions that are found should be as good as possible, given the available amount of computation time. The maximum fitness in consecutive populations should therefore increase as fast as possible. Some techniques for obtaining this result are:

i) injecting heuristic solution into the initial population;
ii) make the initial population large enough so all genetic material is available;
iii) use parallel processing to speed up the generation process;
iv) find optimal values for the mutation probabilities.

As already said in section five, the mapping could also be optimised leading to faster convergence of chromosome fitness.

9 Test results

To evaluate the three schedulers a test case was defined for a difficult scheduling problem. The problem was defined in such a way that not all requests can be scheduled. This causes the scheduler to drop certain tasks, which tasks are dropped can be used for evaluation purposes. In this simulation four sensor functions are defined: search, track, midcourse guidance (MG) and terminal illumination (TI). To make the problem large enough all the tasks have a long duration. A set-up time is required when switching to and from TI representing the time needed to switch from waveform generator in actual MFRs.

The scheduling was done for the single sensor situation and for the duration of one minute. It is assumed that all task requests are made at the beginning of the second in which they should be performed. The problem is illustrated in Figure 4 where the required time for all requested tasks is shown for each second. Since the single sensor problem is used, the available time is, of course, one second.

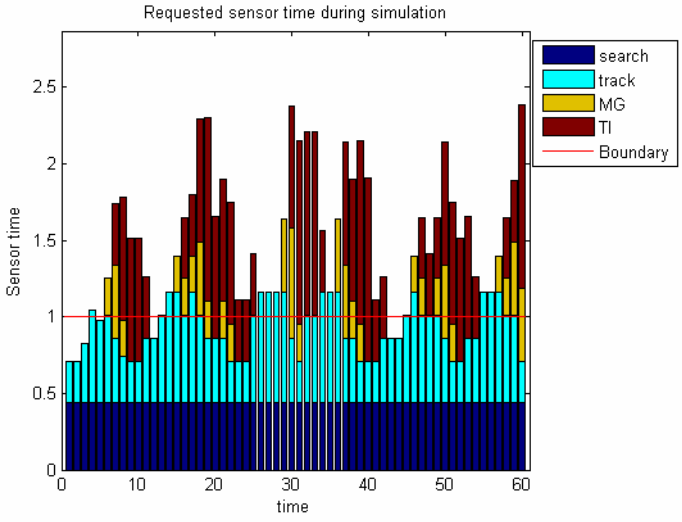

Figure 4 The simulated scheduling problem

To compare the different schedulers we used the utility as given in Eq. (3). This measure for scheduling efficiency was introduced by Thaens [13].

$$U = \frac{\sum_{\forall i} \delta_i P_i}{\sum_{\forall i} P_i} \tag{3}$$

In Eq. (3):

U : Utility;
δ : 1 if task is scheduled correctly, else 0;
P : Priority.

Before the scheduler based on fuzzy Lyapunov could be used for solving the simulation, tests were done to determine values for α and γ. The latter proved to be no factor of importance and was therefore set to 1. The test also showed that $\alpha = 0.8$ was a reasonable value (tests with our GA confirmed this value for Eq. (2) as well). These values are not surprising when looking at Eq. (1) and Eq. (3). The evaluation is based on the priorities of the executed tasks where α reduces the influence of the task duration with respect to the priority. A high value for α is therefore to be expected.

Table 1 Results of four different schedulers

Scheduler	Utility	Dropped budget requests (%)			
		Search	Track	MG	TI
FIFO	*0.5938*	*28.9*	*38.5*	*49.3*	*49.8*
Lyapunov	*0.6805*	*61.4*	*37.7*	*39.6*	*13.4*
GNN	*0.5730*	*19.0*	*42.3*	*47.6*	*46.6*
Online GA	*0.6720*	*48.5*	*41.3*	*49.0*	*10.9*

Since the scheduling problem is highly dependent on time, the utility in time is as important as the average utility. For each of the four schedulers the utility in time is given in Figure 5-8. The average utility in the simulated minute is given in table 1 for each of the three schedulers. For comparison the results of a First-In-First-Out (FIFO) scheduler are also given. In table 1 the percentage of sensor budget that is not executed is also given for each of the four sensor function types.

Looking at the results of the utility, the choice would be to use the scheduler based on Lyapunov synthesis. This scheduler shows a high average utility. The extreme points in time also show that it performs at least as good as the FIFO scheduler when the demand is higher then the available budget. An additional advantage in the military domain is the system predictability: for equal demand, the resulting schedule is the same.

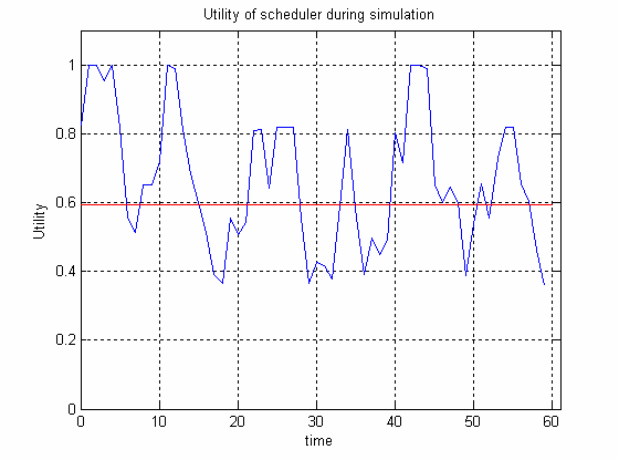

Figure 5 Utility of the FIFO-scheduler

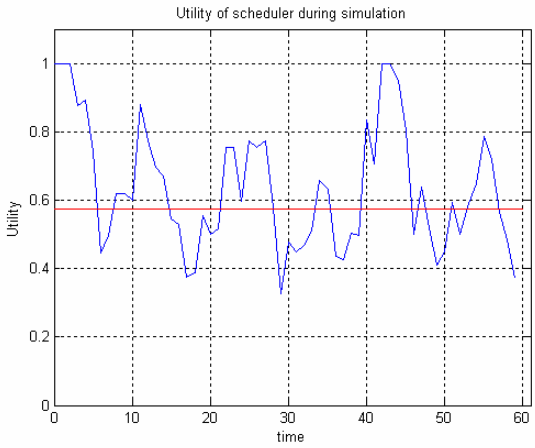

Figure 7 Utility of the GNN scheduler

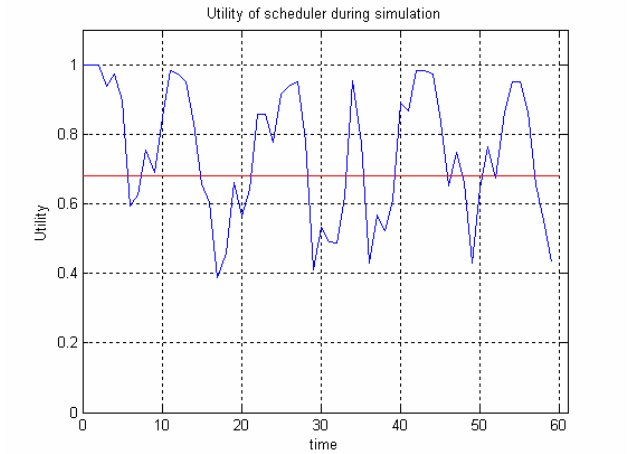

Figure 6 Utility of the fuzzy Lyapunov scheduler

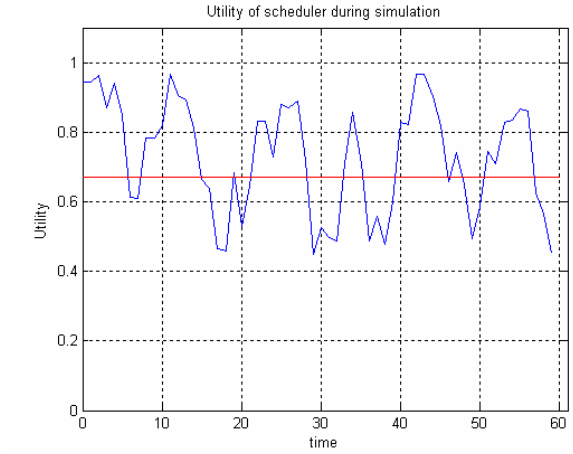

Figure 8 Utility of the online GA for scheduling

A scheduler that is perhaps faster due to fewer computations than the fuzzy Lyapunov based scheduler is the GNN. Our simulation however showed some downsides of this approach. In table 1 we can see that the scheduler drops more TI tasks than search tasks. In practise this means that some of our guided missiles will miss a target. Most likely this effect is due to the NN since the online GA doesn't have this side effect.

The online GA shows promising results. The utility is almost equal to that of the Lyapunov scheduler and it drops less TI tasks. In practise this means that more missiles will hit a target, thus minimising the risk to the mission.

Another advantage of our online GA is that the minimum utility in time is higher when compared to the Lyapunov scheduler. Again this means that it performs slightly better in difficult scheduling situations. This is probably due to the lower percentage of dropped TI tasks. A final remark for all schedulers is the high percentage of dropped MG tasks compared to track tasks. Since the first are of more importance we should consider adjusting our equations somewhat to ensure that MG tasks are dropped less. Furthermore, all three schedulers outperform the FIFO scheduler based on the percentage of dropped tasks.

10 Conclusions

This paper introduced three new scheduling mechanisms for task scheduling in sensor networks. All of those were based on a novel view of the C2 process.

The Lyapunov scheduler proved to be a fast scheduling algorithm that can be implemented relatively easily. The results showed that it can outperform the often used first-in-first-out heuristic. We also suggested that this scheduler can be expanded to accommodate a multi sensor system. An advantage of this scheduler over the others is that it is a predictable method.

Genetic algorithms are a very strong optimisation tool. Using this power in a dedicated parallel process renders it usable in an online scheduler. This results in a high performance scheduler that can be made to be at least as good as any scheduler available today due to the copying without mutation of the fittest chromosomes. The online setting however should be developed and tested further to improve the results.

The disappointing results of NN scheduling are probably due to the complexity of the sensor scheduling problem. Based on the training data no generalisation can be made causing the network to give scheduling results that are worse than the simple FIFO heuristic. A possible solution could be the use of more sophisticated training algorithms that enable the network to generalise better. Since the online implementation seems to work well this however might be unnecessary.

One setback of the GA however remains: a prediction about the optimality of the chosen solution can't be given. Injecting heuristic solutions in the initial population could be a way to guarantee a certain level of optimality.

Further work needs to be done in testing the schedulers in simulation environments. This would show their scheduling efficiency with respect to more realistic task sets. Another important test is to find the saturation points of the schedulers. This boundary indicates the possibility to track a certain number of objects while maintaining some search tasks as well.

Other work that has to be done is to optimise the chromosome mapping in the GA and fine tune its parameters. In Lyapunov some further research could be done on the calculation the buffer weight.

Acknowledgements

This research was conducted at the Delft Cooperation on Intelligent Systems (DECIS) laboratory. This lab is supported by Thales Research and Technology Netherlands, Applied Physics Laboratory (TNO), Delft University of Technology and the University of Amsterdam.

The new approach to the C2 process was developed as part of the STATOR research program supported by Thales Naval Netherlands, the Royal Netherlands Naval College and the International Research Centre for Telecommunications-transmission and Radar of the Delft University of Technology.

References

[1] Alifantis T. and Robinson S., 'Using simulation and neural networks to develop a scheduler advisor', *Proceedings of the 2001 Winter Simulation Conference*

[2] Barbato A., Giustiniani P., 'An improved scheduling algorithm for naval phased array radar', *Alenia Defense Systems, Italy*

[3] Bolderheij F. and Genderen, van P., 'Mission driven sensor management', *7th International Conference on Information Fusion, Stockholm, Sweden, 2004*

[4] Duron C. and Proth J.M., 'Multifunction radar: Task scheduling', *Journal of Mathematical Modelling and Algorithms 1:105-116, 2002*

[5] Huizing A.G. and Bloemen A.A.F., 'An efficient scheduling algorithm for a multifunction radar', *IEEE int. symp. on Phased Array Systems and Technology. 0-7803-3232-6/96, IEEE, 1996*

[6] Jong, de J.L., 'Different representations for the travelling salesman problem using genetic algorithms', *Vrije Universiteit Amsterdam*

[7] Margialot M. and Langholz G., 'Design and analysis of fuzzy schedulers using fuzzy Lyapunov synthesis', *Engineering Applications of Artificial Intelligence, 14:2, p183-188, 2001*

[8] McIntyre G.A., Hintz J.E., 'Sensor measurement scheduling: an enhanced dynamic preemtive algorithm', *Optical Engineering, 37:2, p517-523, 1998*

[9] Mertens B.G.M., 'Reasoning with uncertainty in the situational awareness of air targets', *TNO-FEL 2004, FEL03-S211, TNO The Hague, 2004*

[10] Negnevitsky M., *Artificial Intelligence; A guide to intelligent systems*, Person Education Limited, 2002

[11] Reeves C.R., *Modern heuristic techniques for combinatorial problems*, McGraw-Hill, 1995

[12] Russell S. and Norvig P., *Artificial Intelligence: A Modern Approach 2/E*, Pretence Hall, Pearson Education Inc., 2002

[13] Thaens R., 'Sensor scheduling using intelligent agents', *7th International Conference on Information Fusion, Stockholm, Sweden, 2004*

[14] Thilakawardana S. and Tafazolli R., 'Use of genetic algorithms in efficient scheduling for multi service classes', 2004, http://research.ac.upc.es/EW2004/papers/15.pdf