Resource Allocation with Potts Mean Field Neural Network Techniques

Martin Lagerholm

Department of Theoretical Physics
Lund University

1998

Resource Allocation with Potts Mean Field Neural Network Techniques

Martin Lagerholm
Department of Theoretical Physics
Lund University, Sweden

Thesis for the degree of Doctor of Philosophy

Thesis Advisors: Carsten Peterson and Bo Söderberg
Faculty Opponent: Eric D Mjolsness

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of Lund University, for public criticism in the Auditorium of the Department of Theoretical Physics on May 29 1998, at 10.15 a.m.

DOKUMENTDATABLAD enl SIS 61 41 21

Organization **LUND UNIVERSITY** Department of Theoretical Physics Sölvegatan 14A 223 62 LUND	**Document Name** **DOCTORAL DISSERTATION** **Date of issue** May 1998 **CODEN:** LUNFD6/(NFTF-1037)/1-24 (1998)
Author(s) Martin Lagerholm	**Sponsoring organization**

Title and subtitle

Resource Allocation with Potts Mean Field Neural Network Techniques

Abstract

Potts mean field artificial neural network techniques are developed and applied to airline crew scheduling problems and routing problems. A propagator formalism in terms of Potts neurons is developed to handle global topological issues.

An integrated method for identifying and classifying ECG complexes is presented. Unsupervised self-organizing artificial neural networks are employed to cluster the beats.

Key words

Potts, algorithm, combinatorial optimization, ANN, mean field approximation, routing, unicast, multicast, airline crew scheduling, ECG, NP-complete.

Classification system and/or index terms (if any)

Supplementary bibliographical information		**Language** English
ISSN and key title		**ISBN** 91-628-2933-5
Recipient's notes	**Number of pages** 193	**Price**
	Security classification	

Distribution by (name and address)

Martin Lagerholm, Dept. of theoretical Physics,
Sölvegatan 14 A, SE-223 62 LUND

This thesis is based on the following publications:

I M. Lagerholm, C. Peterson and B. Söderberg
Airline Crew Scheduling with Potts Neurons
Neural Computation **9**, 1589-1599 (1997).

II M. Lagerholm, C. Peterson and B. Söderberg
Airline Crew Scheduling
Using Potts Mean Field Techniques
LU TP 97-10, submitted to *European Journal of Operational Research.*

III M. Lagerholm, C. Peterson and B. Söderberg
Statistical Properties of
Unrestricted Crew Scheduling Problems
LU TP 97-11, submitted to *European Journal of Operational Research.*

IV J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg
A Potts Neuron Approach
to Communication Routing
Neural Computation. **10**, (1998).

V J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg
Local Routing Algorithms
Based on Potts Neural Networks
LU TP 97-5, submitted to *IEEE/ACM Transaction on Networking.*

VI M. Lagerholm, C. Peterson, G. Braccini, L. Edenbrandt and L. Sörnmo
Clustering ECG Using Hermite Functions
and Self-Organizing Maps
LU TP 97-1, submitted to *IEEE Transaction on Biomedical Engineering.*

Preface

This thesis summarizes my scientific work during the stimulating four and a half years I spent in the complex system group at the department of Theoretical Physics in Lund. The main focus (paper I to V) of this thesis is on developing algorithms for finding good approximate solutions to combinatorial optimization problems with global topological constraints. The suggested approach is based on Artificial Neural Network (ANN) techniques and tools from theoretical physics. Paper VI describes a method for classifying real-world ECG complexes. In all papers I have taken an active part in the theory development (especially paper V and VI), paper writing and the simulations behind the presented results were performed by me.

I got attracted to the field of Artificial Neural Networks mainly for two reasons, the first being the *interdisciplinary* nature of the field. ANN techniques can be applied to a broad range of problem types e.g. classification, forecasting, image recognition and optimization, each of these problem types is found in a variety of different fields and applications. The development of the field is also interdisciplinary with contributions from fields such as biology, physiology, statistics, computer science, mathematics and theoretical physics. The second attraction was the *accessibility* to the field for a newcomer. The general ideas are often fairly easy to understand and many of the problems addressed with ANN algorithms are easy to formulate and understand. The bulk of the theoretical work lies in between the formulation of the problem and the solution. My interest in the field has by no means faded, on the contrary, my enthusiasm is now more firmly rooted.

Approaching the ANN field from theoretical physics is very fruitful, tools and concepts such as a Potts neuron, mean field approximation, annealing techniques and propagator formalism adds value and in some cases removes road blocks.

I thank my supervisors Carsten Peterson and Bo Söderberg for their excellent guidance and for all the knowledge they have given me. I thank Mattias Ohlsson, Jari Häkkinen, Hamid Kharraziha and Peter Sutton for proof reading the introduction. Finally, I thank Helena for her love and support.

Introduction

It's a long way to the top,
if you wanna rock 'n' roll.
Young, Young and Scott

A typical human brain is superior to a digital computer in many tasks, e.g. learning, decision making, verbal behavior, visual recognition and motor functions. It is also small, robust and energy efficient. Artificial neural networks (ANNs) stem from simple models of how the neural networks in the brain work. The main reason for most people (including myself) for using and developing ANNs is not to build an artificial brain but to formulate models and algorithms with some of the attractive features of the brain. ANN techniques have proven to be valuable in all the tasks mentioned above even though they in general do not outperform the brain. Today ANN is a computational and programming paradigm.

For hard optimization problems where a rough answer is enough, it seems to me that a human mind doesn't bother whether a problem is formally hard, NP-complete[1], or not, it guesses anyway. But for methods guaranteed to yield the best solution it makes a big difference, for the NP-complete problems the computational demand "explodes" when the size of the problem increases. For the airline crew scheduling and the routing problems (both approached in this thesis) it would, for the large-size end of the probed problems, take a longer time than the age of

[1] An NP-complete problem belongs to a class of problem were no known algorithm could solve it in a time proportional to a polynomial of the problem size. If an exact algorithm is found for one member it could be applied to all.

the universe searching through all possible answers, even if millions of combinations are tried every nanosecond.

The brain's impressive performance still activates a lot of research on general properties of neural networks and will probably continue to do so for a long time. A striking difference in the architecture compared to computers is that the brain hosts of the order 10^{12} (analog) neurons with each neuron connecting approximatively to 1000 other. The belief that the high connectivity is a key to the performance led to a simple model of a neuron which marks the start of the ANN field. This first model of a neuron and some basic concepts used in the papers are described in section 0.1. The sections, 0.2, 0.3 and 0.4 are concerned with the key-tools behind this work, *Mean Field approximation*, *Potts neuron* and a *propagator formalism* respectively. These tools are then applied to form efficient algorithms for airline crew scheduling and routings problems. Section 0.5 gives a brief introduction to the former and to papers I, II and III, section 0.6 is concerned with the latter and introduces papers IV and V. Finally section 0.7 gives a brief background to the detection and clustering of ECG complexes approached in paper VI. All presented papers (except III) make use of a Potts neuron as a basic computational unit, the propagator formalism is utilized in the papers I, II, IV and V.

0.1 Some basic ANN concepts

The very simple model of the brain making the foundation of ANN is based on the traditional viewpoint of the brain as a computer with neurons working as calculators, adding and subtracting signal spikes until a threshold for firing is reached. In 1943 McCulloch and Pitts made a mathematical model of a neuron, the so called McCulloch-Pitts neuron [1], see figure 1 and eq. (1).

$$s_i \to g\left(\sum_j w_{ij} s_j - \theta_i\right). \tag{1}$$

With the transfer function, $g(x)$, chosen to be the Heaviside step func-

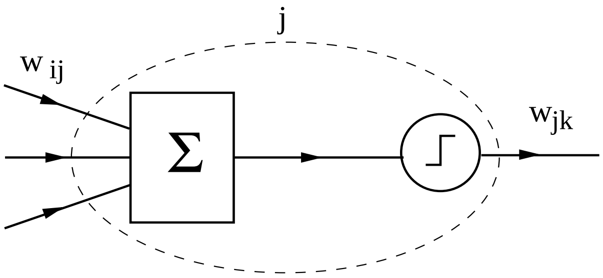

Figure 1: The McCulloch-Pitts neuron. The neuron fires (forwards the value 1) if the weighted input sum $\sum_j w_{ij} s_j \geq \theta_i$.

tion[2], as McCulloch and Pitt did, a neuron either has the value 0 or 1. Other often used transfer functions are $g(x) = sign(x)$ and $g(x) = tanh(x)$. Even though this neuron is simple, a connected network of such devices is capable of performing any computations possible on an ordinary digital computer [1]. This artificial neuron is the basic building block of most types of feed-forward networks, c.f. figure 2. The main application of this architecture is for classification and function approximation tasks.

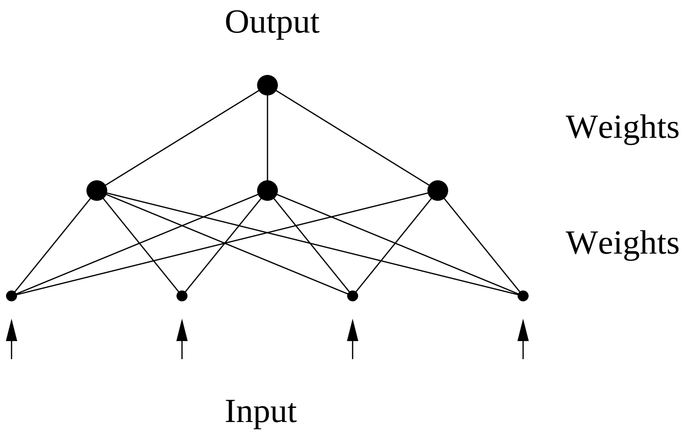

Figure 2: A feed-forward architecture. The large dots represent McCulloch-Pitts neurons and the small marks where the input is made.

[2] $\Theta(x) = 1$ if $x > 0$ otherwise 0.

The input is done at the base and the reaction (output) is received at the top. These kind of networks are trained by feeding inputs to the bottom layer and then adjusting the weights (in a systematic way) so as to get closer to the desired output value. Once trained the weights are frozen and the network is ready to make statements about new inputs.

A multitude of different and important directions within the ANN field have emerged. Two of these directions form the basis of models developed in the paper, i) the Hopfield feed-back model and ii) the Kohonen self-organizing feature extraction architecture. These two types are briefly explained below. I will refrain from giving an overview of other directions since they are not used in any of the papers in this thesis, an overview can be found in e.g. [2].

The Hopfield model

In the beginning of 1980 Hopfield introduced the idea of an energy function into neural network theory (for the associative memory problem). The term energy function originates from an analogy with magnetic spin systems, in other contexts names like Lyapunov function, Hamiltonian, fitness function or objective function would have been used. Somewhat later Hopfield and Tank showed how to apply the ideas to the traveling salesman problem[3] [3] (TSP), this more or less marks the start of using ANN techniques for optimization problems. It is worth noting that the model is expressible in terms of analog circuits with analog amplifiers, hence the algorithm is directly realizable in dedicated hardware. In [3] the TSP was approached by mapping it onto an energy function of the type,

$$E = \text{cost} + \text{constraints}. \tag{2}$$

This energy is expressed in terms of neurons in such a way that the minimum corresponds to a neuron configuration representing a solution to the problem. As an example of how to map a problem into such an energy function the graph bisection problem will be used, see figure 3. The task is to divide a given graph (nodes and links) into two equal parts with a minimal cut size (number of links) between them. For each node i a binary neuron, s_i, is assigned, representing whether the neuron

[3]A closed tour between a given set of cities should be performed with minimum traveling.

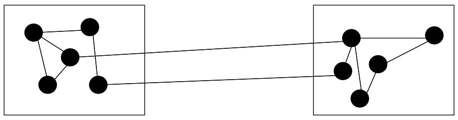

Figure 3: A graph bisection problem.

is in the left box, $s_i = +1$ or in the right box, $s_i = -1$. For each pair of nodes let $w_{ij} = 1$ if a link between them exists and otherwise 0. A suitable energy function is

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j + \frac{\alpha}{2}\left(\sum_i s_i\right)^2 . \tag{3}$$

The first term expresses the cut size cost whereas the second term is a constraint term, penalizing situations where the nodes are not equally partitioned, α sets the relative strength between the two terms. The cost and the constraint have to "compete", i.e the problem is frustrated, this typically leads to the appearance of many local minima.

Once the energy is written down the next step is to find the minimum. This could be done by using e.g. mean field methods (see section 0.3).

Self-organizing maps

The Kohonen architecture, see figure 4, is used to cluster data in an unsupervised manner. The main application is for reducing the dimensionality of data, by mapping it onto a two dimensional lattice of nodes. The network works as a black box — fed only with the set of input patterns (vectors) it groups together those with similar features by associating each input vector with the closest node. The nodes are moved towards cluster centers by iteratively repeating: For each input pattern move the closest (in the input space) node a little bit closer to the position of the input. The closest node is called the winner. With a suitable initial position of the nodes, this procedure will result in a clustering; the set of input vectors sharing the same winner defines a cluster. By modifying the update strategy neighbouring units in the 2 dimensional output map can be made to correspond to adjacent clusters in the input space, this can be accomplished by dragging the neighbouring (in the output space) nodes along when a node is moved, for details see paper

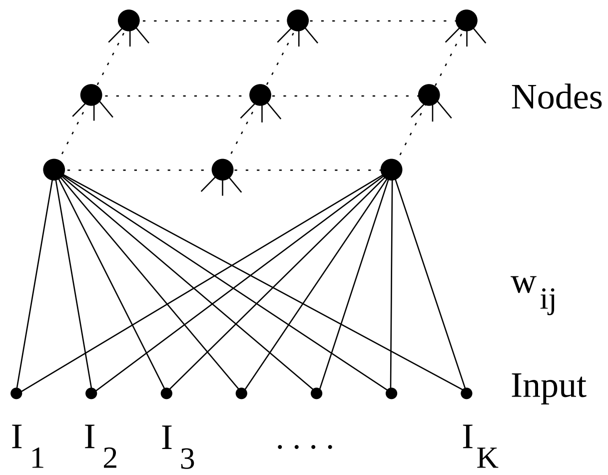

Figure 4: The Kohonen self-organizing feature extraction architecture. The nodes' positions in the 2-dimensional output space are fixed. Their positions in the K-dimensional input space, defined by $\vec{w}_i$ for each output node i, are changed according to the update procedure described in the text.

VI. For a general description of self-organizing maps see [4].

To fully utilize the potential of ANN techniques dedicated hardware is needed. Due to the rapid evolution of the field and models, hardware implementations lag behind, and the models, as is the case in this thesis, must therefore be simulated on ordinary computers.

0.2 Potts Neurons

It is advantageous to use a multi-valued neuron, Potts neuron, instead of a binary neuron for many types of problems, e.g. the airline crew scheduling problem and routing problems. Using a Potts neurons when encoding a problem can reduce the number of terms needed in the energy function (of Hopfield type eq. (2)). Fewer energy terms in general result in a less complex energy landscape and hence the global minimum will be easier to locate. The general graph partition problem will be used as an example, here the task is to evenly partition N nodes into K boxes with a minimal cut-size between them, c.f. figure 5. If binary neurons

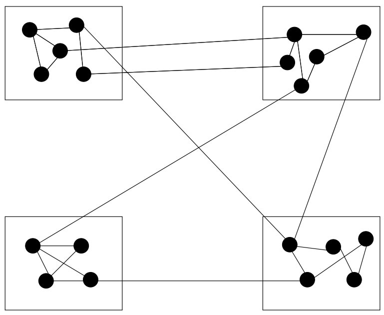

Figure 5: A graph partition problem.

were to be used (as was suitable for the graph-**bi**section problem), then a neuron, s_{ia}, could encode whether node i is in box a or not. Since a neuron only can be in one box (in the final result), an energy term enforcing

$$\sum_b s_{ib} = 1 \tag{4}$$

must be used. A more efficient encoding reducing the available solution space is to use Potts neurons. A Potts neuron has K possible values, $\vec{s}_i = (s_{i1}, s_{i2}, ..., s_{iK})$ where the j:th possible state is given by $s_{ij} = 1$ and $s_{ik} = 0$ for $k \neq j$. These states are then mutually orthogonal and automatically fulfill eq. (4). A Potts neuron could then encode to which box a neuron is assigned, e.g.

$$\vec{s}_i = \{0\ 0\ 1\ 0\} \tag{5}$$

means that node i is in box 3.

0.3 The Mean-Field Approximation

Given a problem mapped onto an energy function $E(\mathbf{s})$ (of Hopfield type) the next step is to locate a global minimum. A gradient descent method would lead to the nearest minimum which probably is not a global minimum. Instead a stochastic method like simulated annealing[4] [5] allowing for uphill moves could be applied. This technique is very CPU consuming. The simulated annealing equations could be replaced with deterministic equations, by using a mean field approximation, replacing the sharp Potts neuron components $\vec{s}_i$ with a mean value at a given (artificial) temperature T,

$$\vec{v}_i = \langle \vec{s}_i \rangle_T. \tag{6}$$

Starting from the Boltzmann energy distribution and rewriting the expression and keeping the main contribution has the result that the j:th component of Potts neuron $\vec{v}_i$ has the value

$$v_{ij} = \frac{e^{u_{ij}/T}}{\sum_k e^{u_{ik}/T}} \tag{7}$$

where u_{ij} is the "local field",

$$u_{ij} = -\frac{\partial E(\mathbf{v})}{\partial v_{ij}} \tag{8}$$

The steps for arriving at this result and a more elaborate explanation of the Potts mean field technique is found in Paper II.

This approach, using Potts mean field neurons for mapping optimization problems onto ANN was suggested in 1989 by Peterson and Söderberg who successfully applied it to the graph partition problem and the traveling salesman problem [6].

The basic steps for arriving at an approximative algorithm for an optimization problem using the Potts mean-field technique are:

- Map the problem onto a Hopfield type of energy function, using Potts neurons.
- Start at a high temperature with the neurons close (with a small random variation) to the high T limit of eq. (7).

[4]By analogy with real annealing e.g. a congealing liquid.

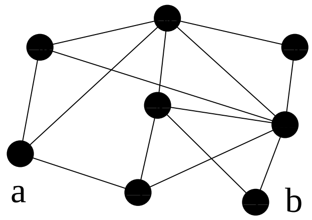

Figure 6: A signal network composed of nodes and links.

- While lowering the temperature, annealing, iteratively calculate the local field eq. (8) and update the neuron values according to eq. (7).
- When the neurons have converged, an answer to the problem is just to be decoded from the neuron values. If this suggested solution does not obey the constraints or, if it is too far from the global minimum, the parameters (like α in eq. (3)) might have to be tuned and/or the annealing be done more carefully.

Since the energy is only used to derive the local field, one could replace the first step by instead formulating cost and constraints directly in terms of the local field,

$$u = \text{cost term} + \text{constraint terms} \tag{9}$$

This is especially useful for terms expressed using a propagator (see below) since these are often difficult, and from an algorithmic viewpoint unnecessary, to express in terms of an energy.

0.4 The Propagator

When topological constraints have to be formulated a propagator formalism can be very helpful. To illustrate this a simple signal network will be used, see figure 6. Let each node, except b, be associated with a Potts neuron that encodes which outgoing link an incoming signal should use, $v_{ij} = 1$ if signals from i are passed on to j and otherwise

0. If node a sends a signal that should be received by node b then b must be a sink since it should not pass on any signal, i.e. $v_{bj} = 0$ for all j (thus it is not a Potts neuron since it does not obey eq. (4). Using the mean field approximation, at a non-zero temperature, means that the states of the Potts neurons are not sharp (0/1), the values are rather conveniently interpreted as probabilities ($0 \leq v_{ij} \leq 1$). This in turn means that questions like (i) "How much of the signal from node a passes node j on its way to b?" and (ii) "How many links are used in the path a to b?" are much harder to answer. Noting that the signal from a eventually arrives at b since this is the only sink in the system, the answer to the first question can be derived by systematically going through all possible paths from a to j. Starting with "Am I already at j?", δ_{aj}, then check how "much" the direct connection is used, v_{aj} (zero if no link exists), continuing by adding contributions from paths going via one other neuron, $\sum_k v_{ak}v_{kj}$ etc., one finds the answer to question (i):

$$P_{aj} = \delta_{aj} + v_{aj} + \sum_k v_{ak}v_{kj} + \sum_{kl} v_{ak}v_{kl}v_{lj} + \sum_{klm} v_{ak}v_{kl}v_{lm}v_{mj} + \ldots \quad (10)$$

This is an element of a *propagator* matrix defined as (since eq. (10) is a geometrical sum)

$$\mathbf{P} = (\mathbf{1} - \mathbf{v})^{-1}. \quad (11)$$

All the elements of the propagator are calculated by simply inverting a matrix! The answer to the second question could also be expressed using the propagator: the number of links used in the connection from a to b is

$$(\mathbf{P}^2)_{ab} - P_{ab}. \quad (12)$$

This formalism is very powerful and will be used when formulating algorithms for the airline crew scheduling problem and routing problems. For details and more examples on the usefulness of this propagator formalism I recommend the interested reader to study paper II.

0.5 Airline Crew Scheduling

The resource planning of an airline company is a very complex problem, usually it is divided into four main blocks which then are solved more or less sequentially.
Timetable construction: Construct a timetable, optimized with respect to market, available time slots at airports etc.
Fleet assignment: Assign a given aircraft fleet to the timetable such that a revenue, depending on size of the aircraft, fuel consumption, staff requirements, etc., is maximized. An example of a constraint is that all types of aircraft are not allowed to land on all airports.
Crew scheduling: A crew should be assigned to each flight obeying a large number of governmental regulations, union demands and collective agreements. This should be done so that the total expense for the crews is minimized (utilizing the crew effectively).
Crew Assignment: Assigning persons to the different crews taking things as vacations and training into account.
For a more complete problem description see e.g. [7].

Next to fuel costs the crew is the largest running cost. Improving the crew scheduling a few percent would lead to millions of dollars of saving for a typical major airline company.

In what follows the focus is on the crew scheduling problem. The "standard" approach is to split the problem into two parts: (1) generating a large pool of rotations[5], and (2) seeking the best subset of those that cover all flights (a set partition problem). Part (2) is often done with linear programming techniques. Another approach is to start from an initial and crude solution and then proceed by fine-tuning sub parts in an iterative manner, both these techniques are described in [7].

[5] A sequence of flights starting and ending at the home-base (hub).

Paper I
The approach to the crew scheduling problem developed in this letter is quite different from the standard methods, we do not rely on generating rotations, the optimization is done in one shot. The problem is mapped onto an ANN using Potts mean field neurons. We limit ourselves to two crucial and difficult constraints, bounds on the total working time and the number of take-offs per rotation respectively. In this paper the propagator is a novel key ingredient for handling the constraints. The algorithm is tested on artificial random problems tuned to resemble real-world conditions. Very good results are obtained for a variety of problem sizes. Realistic problem sizes are typically solved within a few minutes, partly due to a prior reduction of the problem size, based on an analysis of the local arrival/departure structure at single airports.

Paper II
The algorithm and the tools used in Paper I are explained and derived in more detail. A small toy example is used to illustrate the reduction technique, the encoding of the problem and the use of the propagator.

Paper III
This paper is entirely devoted to a statistical analysis of the unrestricted crew scheduling problem. By requiring a solution at the minimal energy, airports could effectively be split into smaller parts. This increases the number of effective airports and therefore decrease the number of possible combinations of flights. Typically this procedure reduces the state space by a factor $\sim 2^{N_a}$, where N_a is the (original) number of airports. Some basic aspects of this analysis were developed and used for the reduction technique in paper I and II.

The next step to test this Potts mean-field approach would be to include all the constraints used in a real-world airline crew scheduling problem.

0.6 Routing

The following is a routing problem:
Given a set of cities linked together with a network of roads, find the fastest path between two cities. The quality of the roads sets a maximum speed (possibly not the same in both directions of a road).
In the late 50's an algorithm to solve this problem was proposed by Bellman [8], it will be referred to as **BF** (Bellman-Ford). This algorithm is rapid, the needed number of iterations scales as the product of the number of cities and roads.

Today most activity on routing problems concerns telecommunication applications (e.g. the Internet), in their language the problem above is called a *unicast* problem. Cities translate to nodes, roads to arcs (links) and an arc cost is used instead of road quality. The routing decision system in the early Internet was based on the BF algorithm. Today it is still important in many protocols and applications. For a general overview of protocols and techniques used in data networks see [9].

An arc has limited capacity, hence as soon as more than one unicast is to be routed this constraint come into play and makes the problem much harder. In the case of competition for an arc, the signal that is globally least expensive to force to a detour should change route, but this depends on how all the other signals are routed and how expensive they are to "move", so settling this competition could start an "avalanche" of calculations. In the *multiple unicast* problem the task is to assign paths between many pairs of nodes simultaneously without exceeding the capacity of any arc.

Paper IV
The BF algorithm is recast in a Potts mean field language and then extended to suite the multiple unicast problem (multiple shortest path problem). The propagator formalism developed in papers I and II is a crucial tool for handling the capacity constraint and other topological issues. The method is evaluated for synthetic problems of varying sizes and load levels by comparing to exact solutions from a branch-and-bound method and to approximate solutions from simple heuristics. With very few exceptions, the Potts approach gives legal solutions of high quality.

Another interesting and difficult routing problem is the *multicast* problem. The task is to send one and the same message to several receivers with a minimal cost of the resulting tree (sum of the individual arc costs). The multicast problem appears in many "new" applications e.g. software distribution, mailing lists, push-techniques and Internet radio/TV.

In the *multiple multicast* problem, several single multicasts should be routed simultaneously. Just like in the multiple unicast problem the limited arc capacity adds the complication that one signal could block the others, forcing them to take expensive detours or even to fail.

Paper V
In this paper algorithms for the multiple unicast, multicast and multiple multicast are presented. The basic toolbox is the mean field approximation, Potts neurons and the propagator formalism.

A video conference where all participants send images/messages to each other is a special case of the multiple multicast problem. Video conferences over the Internet have become possible just in the recent years, previously very few people had the computer memory, hardware, bandwidth and CPU power necessary to manipulate the large amount of digital pictures and sound. So my guess is that the interest of this routing problem will increase in the near future.

The next development step of this approach would be to extend the algorithms to handle dynamical situations in real time.

0.7 ECG classification

Visual analysis of electrocardiograms (ECGs) is tedious and operator dependent. Automated systems for detection of arrhythmias considerably reduce the amount of time the operator needs to spend. Several commercial systems are available for ECG analysis. However, their performances deteriorate markedly when noise and artefacts are present which is typical for long-term[6] ECGs.

In figure 7 the Q, R and S peaks, a QRS complex, of a typical normal beat are depicted.

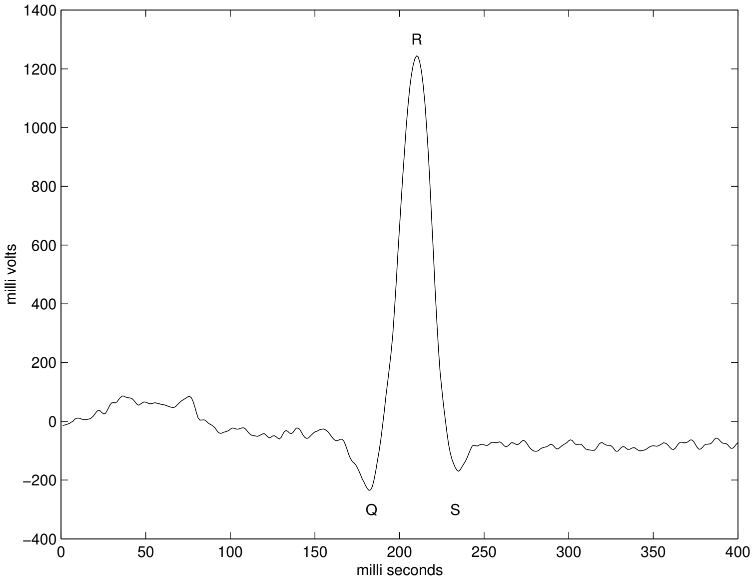

Figure 7: A typical QRS complex, the Q, R and S peaks are marked.

[6]The raw signal from e.g. a tape recorder carried by a patient during an ordinary day, this is in sharp contrast to rest-ECGs which often are much "cleaner" and furthermore averaged over several beats.

Paper VI
An integrated method for identifying and classifying QRS complexes is presented. Each complex is decomposed into Hermite basis functions, from which the resulting coefficients and widths are used to represent the beat. We use a Kohonen network for clustering the beats into a fixed number of clusters. A Potts neuron formalism is used to steer similar clusters to be presented as neighbours in the output map. The performance of this method is good, it outperforms a published supervised learning method using the same data as well as a more conventional template cross-correlation clustering method.

By adding a few tools and constructing a graphical user interface I believe that the method presented in paper IV could be useful for a cardiologist. For example, the user could have the option of fine tuning the resulting map by moving individual beats between clusters. Another useful tool would be the possibility to break clusters into new sub-clusters.

Bibliography

[1] W.S. McCulloch and W.H. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", Bull. Math, Biophys. **5**, 115 (1943).

[2] J. Hertz, A. Krogh and R. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, California (1991).

[3] J.J. Hopfield and D.W. Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* **52**, 141 (1985).

[4] T. Kohonen, *Self-Organizing Maps*, Springer-Verlag, Berlin, Heidelberg (1997).

[5] S. Kirkpatrick, C.D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing", *Science* **220**, 671 (1983).

[6] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[7] Edited by Gang Yu *Operations Research in the Airline Industry* (International Series in Operations Research & Management Science, 9), Kluwer Academic Publishers, USA (1998).

[8] R. Bellman, "On a Routing Problem", *Quarterly of Applied Mathematics* **16** 87 (1958).

[9] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, New Jersey (1987).

Chapter 1

Airline Crew Scheduling with Potts Neurons

Paper I

LU TP 96-6

Airline Crew Scheduling with Potts Neurons

Martin Lagerholm[1], Carsten Peterson[2]
and Bo Söderberg[3]

Department of Theoretical Physics, University of Lund
Sölvegatan 14A, S-223 62 Lund, Sweden

Neural Computation **9**, 1589-1599 (1997)

Abstract:
A Potts feedback neural network approach for finding good solutions to resource allocation problems with a non-fixed topology is presented. As a target application the airline crew scheduling problem is chosen. The topological complication is handled by means of a propagator defined in terms of Potts neurons. The approach is tested on artificial random problems tuned to resemble real-world conditions. Very good results are obtained for a variety of problem sizes. The computer time demand for the approach only grows like (number of flights)^3. A realistic problem typically is solved within minutes, partly due to a prior reduction of the problem size, based on an analysis of the local arrival/departure structure at the single airports.

[1]martin@thep.lu.se
[2]carsten@thep.lu.se
[3]bs@thep.lu.se

1.1 Introduction

Feedback neural network have in the last decade emerged as a useful method to obtain good approximate solutions to various resource allocation problems [1, 2, 3, 4]. Most applications have concerned fairly academic problems like the traveling salesman problem (TSP) and various graph partition problems [1, 2]. In refs. [3, 4] high school scheduling was approached. The typical approach proceeds in two steps: (1) map the problem onto a neural network (spin) system with a problem-specific energy function, and (2) minimize the energy by means of deterministic mean field (MF) equations, which allow for a probabilistic interpretation. Two basic mapping variants are common: a hybrid (template) approach [5], and a "purely" neural one. The template approach is advantageous e.g. for low-dimensional geometrical problems like the TSP, whereas for generic resource allocation problems, a purely neural Potts encoding is preferable.

A very challenging resource allocation problem is airline crew scheduling, where a given flight schedule is to be covered by a set of crew *rotations*, each consisting in a connected sequence of flights (*legs*), starting and ending at a given home base (*hub*). The total crew waiting time is then to be minimized, subject to a number of restrictions on the rotations. This application differs strongly from e.g. high school scheduling [3, 4] in the existence of non-trivial topological restrictions. A similar structure occurs in multi-task phone routing.

A common approach to this problem consists in converting it into a *set covering problem*, by (1) generating a large pool of legal rotation templates, and (2) seeking a subset of the templates that precisely covers the entire flight schedule. Solutions to the set covering problem are then found with linear programming techniques or feedback neural network methods [6]. A disadvantage with this method is that the rotation generation for computational reasons has to be non-exhaustive for a large problem; thus, only a fraction of the solution space is available.

The approach to the crew scheduling problem developed in this letter is quite different, and proceeds in two steps. First, the full solution space is narrowed down using a reduction technique that removes a large part of the sub-optimal solutions. Then, a mean field annealing approch based on Potts neurons is applied, where a novel key ingredient is the use of a propagator formalism for handling topology, leg-counting, etc.

The method, which is explored on random artificial problems resembling real-world situations, performs well with respect to quality, with a computational requirement that grows like N_f^3, where N_f is the number of flights.

1.2 Nature of Problem

Typically, a real-world flight schedule has a basic period of one week. Given such a schedule in terms of a set of N_f weekly flights, with specified times and airports of departure and arrival, a crew is to be assigned to each flight such that the total crew waiting time is minimized, subject to the constraints:

- Each crew must follow a connected path – a *rotation* – starting and ending at the hub (see fig. 1.1).
- The number of flight legs in a rotation must not exceed a given upper bound $L_{\max}$.
- The total duration (flight + waiting time) of a rotation is similarly bounded by $T_{\max}$.

These are the crucial and difficult constraints; in a real-world problem there are often some 20 additional ones, which we for simplicity neglect; they constitute no additional challenge from an algorithmic point of view.

Without the above constraints, the problem would reduce to that of minimizing waiting times independently at each airport – the *local problems*; these can be solved exactly in polynomial time, e.g. by pairwise connecting arrival flights with subsequent departure flights. It is the global structural requirements that make the crew scheduling problem a challenge.

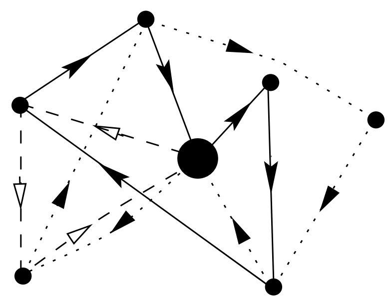

Figure 1.1: Schematic view of the 3 crew rotations starting and ending in a hub.

1.3 Reduction of Problem Size – Airport Fragmentation

Prior to developing our artificial neural network method, we will describe a technique to reduce the size of problem, based on the local flight structure at each airport.

With the waiting time between an arriving flight i and a departing flight j defined as

$$t_{ij}^{(\mathrm{w})} = \left(t_j^{(\mathrm{dep})} - t_i^{(\mathrm{arr})}\right) \bmod \text{period}, \tag{1.1}$$

the total waiting time for a given problem can only change by an integer times the period. By demanding a minimal waiting time, the local problem at an airport typically can be split up into independent subproblems, where each contains a subset of the arrivals and an equally large subset of the departures. For example, with A/D denoting arrival/departure, the time ordering $[AADDAD]$ can, if minimal waiting time is demanded, be divided into two subproblems, $[AADD]$ and $[AD]$. Some of these are trivial ($[AD]$), forcing the crew of an arrival to continue to a particular departure. The minimal total wait-time for the local problems is straight-forward to compute, and will be denoted by $T_{\min}^{\text{wait}}$.

Similarly, by demanding a solution (assuming one exists) with $T_{\min}^{\text{wait}}$ also for the constrained global problem, this can be reduced as follows:

- *Airport fragmentation*: Divide each airport into *effective airports* corresponding to the non-trivial local subproblems.
- *Flight clustering*: Join every forced sequence of flights into one effective *composite flight*, which will thus represent more than one leg and have a formal duration defined as the sum of the durations of its legs and the waiting times between them.

The *reduced problem* thus obtained differs from the original problem only in an essential reduction of the sub-optimal part of the solution space; the part with minimal waiting time is unaffected by the reduction. The resulting information gain, taken as the natural logarithm of the decrease in the number of possible configurations, empirically seems to scale approximately like $1.5\times$ (number of flights), and ranges from 100 to 2000 for the problems probed.

The reduced problem may in most cases be further separated into a set of independent *subproblems*, that can be solved one by one. Some of the composite flights will formally arrive at the same effective airport they started from. This does not pose a problem. Indeed, if the airport in question is the hub, such a single flight constitutes a separate (trivial) subproblem, representing an entire forced rotation. Typically, one of the subproblems will be much larger than the rest, and will be referred to as the *kernel problem*, while the remaining subproblems will be essentially trivial.

In the formalism below, we allow for the possibility that the problem to be solved has been reduced as described above, which means that flights may be composite.

1.4 Potts Encoding

A naive way to encode the crew scheduling problem would be to introduce Potts spins in analogy with what was done in refs. [3, 4] where each event (lecture) is mapped onto a resource unit (lecture-room, time-slot). This would require a Potts spin for each flight to handle the mapping onto crews.

Since the problem consists in linking together sequences of (composite) flights such that proper rotations are formed, starting and ending at the hub, it appears more natural to choose an encoding where each flight i is mapped, via a Potts spin, onto the flight j to follow it in the rotation:

$$s_{ij} = \begin{cases} 1 & \text{if flight } i \text{ precedes flight } j \text{ in a rotation} \\ 0 & \text{otherwise} \end{cases}$$

where it is understood that j be restricted to depart from the (effective) airport where i arrives. In order to ensure that proper rotations are formed, each flight has to be mapped onto precisely one other flight. This restriction is inherent in the Potts spin, defined to have precisely one component "on":

$$\sum_j s_{ij} = 1 \tag{1.2}$$

In order to start and terminate rotations, two auxiliary flights are introduced at the hub: a dummy arrival a, forced to be mapped onto every proper hub departure, and a dummy departure b, to which every proper hub arrival must be mapped. No other mappings are allowed at the hub. Thus, a acts as a source, and b as a sink of rotations; every rotation starts with a and terminates with b. Both have zero duration and leg-count, as well as associated wait-times.

Global topological properties, leg-counts and durations of rotations, etc., cannot be described in a simple way by polynomial functions of the spins. Instead, they are conveniently handled by means of a *propagator* matrix, $\mathbf{P}$, derived from the Potts spin matrix $\mathbf{s}$ (including a, b components) as

$$\begin{aligned} P_{ij} &= \left((\mathbf{1}-\mathbf{s})^{-1}\right)_{ij} = \delta_{ij} + s_{ij} + \sum_k s_{ik}s_{kj} \\ &+ \sum_{kl} s_{ik}s_{kl}s_{lj} + \sum_{klm} s_{ik}s_{kl}s_{lm}s_{mj} + \ldots \end{aligned} \tag{1.3}$$

A pictorial expansion of the propagator is shown in fig. 1.2. The interpretation is obvious: P_{ij} counts the number of connecting paths from flight i to j. Similarly, an element of the matrix square of P,

$$\sum_k P_{ik}P_{kj} = \delta_{ij} + 2s_{ij} + 3\sum_k s_{ik}s_{kj} + \ldots \tag{1.4}$$

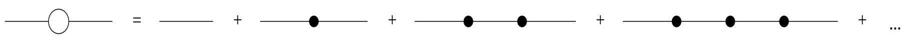

Figure 1.2: Expansion of the propagator P_{ij} ($\bigcirc$) in terms of s_{ij}. A line represents a flight, and ($\bullet$) a touch-down.

counts the total number of (composite) legs in the connecting paths, while the number of proper legs is given by

$$\sum_k P_{ik}L_kP_{kj} = L_i\delta_{ij} + s_{ij}\left(L_i + L_j\right) + \sum_k s_{ik}s_{kj}\left(L_i + L_k + L_j\right) + \ldots \tag{1.5}$$

where L_k is the intrinsic number of single legs in the composite flight k. Thus, $L_{ij} \equiv \sum_k P_{ik}L_kP_{kj}/P_{ij}$ gives the *average leg count* of the connecting paths. Similarly, the *average duration* (flight + waiting time) of the paths from i to j amounts to

$$T_{ij} \equiv \frac{\sum_k P_{ik}t_k^{(\mathrm{f})}P_{kj} + \sum_{kl} P_{ik}t_{kl}^{(\mathrm{w})}s_{kl}P_{lj}}{P_{ij}} \tag{1.6}$$

where $t_i^{(\mathrm{f})}$ denotes the duration of the composite flight i, including the embedded waiting time. Furthermore, any closed loop (such as obtained e.g. if two flights are mapped onto each other) will make $\mathbf{P}$ singular – for a proper set of rotations, $\det\mathbf{P} = 1$.

The problem can now be formulatet as follows. Minimize[4] T^{ab} subject to the contraints:

$$L_{ib} \leq L_{max} \tag{1.7}$$
$$T_{ib} \leq T_{max} \tag{1.8}$$

for every departure i at the hub.

1.5 Mean Field Approach

We use a *mean field* (MF) annealing approach in the search for the global minimum. The discrete Potts variables, s_{ij}, are replaced by continuous

[4]Minimizing this minimizes the total wait time, since the total flight time is fixed.

MF Potts *neurons*, v_{ij}. They represent thermal averages $< s_{ij} >_T$, with T an artificial temperature to be slowly decreased (annealed), and have an obvious interpretation of probabilities (for flight i to be followed by j). The corresponding probabilistic propagator $\mathbf{P}$ will be defined as the matrix inverse of $\mathbf{1} - \mathbf{v}$.

The neurons are updated by iterating the MF equations

$$v_{ij} = \frac{\exp(u_{ij}/T)}{\sum_k \exp(u_{ik}/T)} \tag{1.9}$$

for one flight i at a time, by first zeroing the i:th row of $\mathbf{v}$[5], and then computing the relevant *local fields* u_{ij} entering eq. (1.9) as

$$\begin{aligned} u_{ij} &= -c_1 t_{ij}^{(\mathrm{w})} - c_2 P_{ji} - c_3 \sum_k v_{kj} \\ &- c_4 \Psi\left(T_{\mathrm{rot}}^{(\mathrm{ij})} - T_{\max}\right) - c_5 \Psi\left(L_{\mathrm{rot}}^{(\mathrm{ij})} - L_{\max}\right) \end{aligned} \tag{1.10}$$

where j is restricted to be a possible continuation flight to i. In the first term, $t_{ij}^{(\mathrm{w})}$ is the waiting time between flight i and j. The second term suppresses closed loops, and the third term is a soft exclusion term, penalizing solutions where two flights point to the same next flight. In the fourth and fifth terms, $L_{\mathrm{rot}}^{(\mathrm{ij})}$ stands for the total leg count and $T_{\mathrm{rot}}^{(\mathrm{ij})}$ for the duration of the rotation if i where to be mapped onto j, and amount to

$$L_{\mathrm{rot}}^{(\mathrm{ij})} = L_{ai} + L_{jb} \tag{1.11}$$

$$T_{\mathrm{rot}}^{(\mathrm{ij})} = T_{ai} + t_{ij}^{(\mathrm{w})} + T_{jb} \tag{1.12}$$

The penalty function Ψ, used to enforce the inequality constraints [7, 6], is defined by $\Psi(x) = x\Theta(x)$ where Θ is the Heaviside step function. Normally, the local fields u_{ij} are derived from a suitable energy function; however, for reasons of simplicity, some of the terms in eq. (1.10) are chosen in a more pragmatic way.

After an initial computation of the propagator $\mathbf{P}$ from scratch, it is subsequently updated according to the Sherman-Morrison algorithm for

[5] To eliminate self-couplings

incremental matrix inversion [8]. An update of the i:th row of $\mathbf{v}$, $v_{ij} \to v_{ij} + \delta_j$, generates precisely the following change in the propagator $\mathbf{P}$:

$$P_{kl} \quad \to \quad P_{kl} + \frac{P_{ki}\, z_l}{1 - z_i} \tag{1.13}$$

$$\text{where} \quad z_l \quad = \quad \sum_j \delta_j\, P_{jl} \tag{1.14}$$

Inverting the matrix from scratch would take $O(N^3)$ operations, while the (exact) scheme devised above only requires $O(N^2)$ per row.

As the temperature goes to zero, a solution crystallizes in a winner-takes-all dynamics: for each flight i, the largest u_{ij} determines the continuation flight j to be chosen.

1.6 Test Problems

In choosing test problems our aim has been to maintain a reasonable degree of realism, while avoiding unnecessary complication and at the same time not limiting ourselves to a few real-world problems, where one can always tune parameters and procedures to get good performance. In order to accomplish this we have analyzed two typical real-world template problems obtained from a major airline: one consisting of long distance (LD), the other of short/medium distance (SMD) flights. As can be seen from fig. 1.3, LD flight time distributions are centered around long times, with a small hump for shorter times representing local continuations of long flights. The SMD flight times have a more compact distribution.

For each template we have made a distinct problem generator producing random problems resembling the template. A problem with a specified number of airports and flights is generated as follows: First, the distances (flight-times) between airports are chosen randomly from a suitable distribution. Then, a flight schedule is built up in the form of legal rotations starting and ending at the hub. For every new leg, the waiting time and the next airport are randomly chosen in a way designed to make the resulting problems statistically resemble the respective template problem.

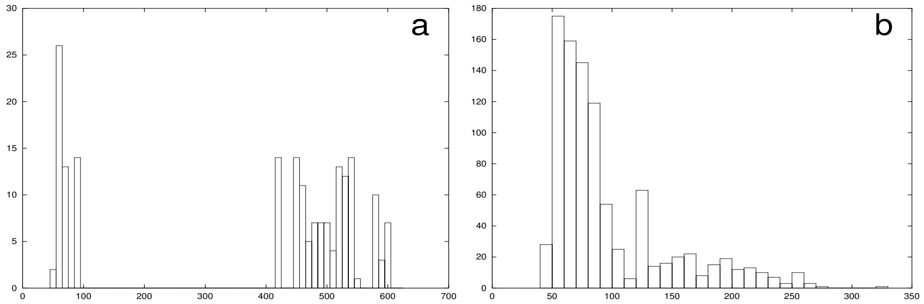

Figure 1.3: Fligh time distributions in minutes for **(a)** LD and **(b)** SMD template problems.

Due to the excessive time consumption of the available exact methods, e.g. branch and bound, the performance of the Potts approach cannot be tested against these – except for in this context ridiculously small problems, for which the Potts solution quality matches that of an exact algorithm. For artificial problems of more realistic size we circumvent this obstacle in the following way: since problems are generated by producing a legal set of rotations, we add in the generator a final check that the implied solution yields $T^{\rm wait}_{\rm min}$; if not, a new problem is generated. Theoretically, this might introduce a bias in the problem ensemble; empirically, however, no problems have had to be redone. Also the two real problems turn out to be solvable at $T^{\rm wait}_{\rm min}$. Each problem then is reduced as described above (using a negligible amount of computer time), and the kernel problem is stored as a list of flights, with all traces of the generating rotations removed.

1.7 Results

We have tested the performance of the Potts MF approach for both LD and SMD kernel problems of varying sizes. As an annealing schedule for the serial updating of the MF eqs. (1.9,1.13), we have used $T_n{=}kT_{n-1}$ with $k = 0.9$. In principle, a proper value for the initial temperature T_0 can be estimated from linearizing the dynamics of the MF equations. We have chosen a more pragmatic approach: The initial temperature is assigned a tentative value of 1.0, which is dynamically adjusted based on the measured rate of change of the neurons until a proper T_0 is found. The values used for the coefficients c_i in eq. (1.10) are chosen in an equally simple and pragmatic way: $c_1 = 1/\text{period}$, $c_2 = c_3 = 1$, while $c_4 = 1/ < T^{\text{rot}} >$ and $c_5 = 1/ < L^{\text{rot}} >$, where $< T^{\text{rot}} >$ is the average duration (based on $T^{\text{wait}}_{\text{min}}$) and $< L^{\text{rot}} >$ the average leg count per rotation, both of which can be computed beforehand. It is worth stressing that these parameter settings have been used for the entire range of problem sizes probed.

When evaluating a solution obtained with the Potts approach, a check is done as to whether it is legal (if not, a simple post-processor restores legality – this is only occasionally needed), then the solution quality is probed by measuring the excess waiting time R,

$$R = \frac{T^{\text{wait}}_{\text{Potts}} - T^{\text{wait}}_{\text{min}}}{\text{period}}, \tag{1.15}$$

which is a non-negative integer for a legal solution.

For a given problem size, as given by the desired number of airports N_a and flights N_f, a set of 10 distinct problems is generated. Each problem is subsequently reduced, and the Potts algorithm is applied to the resulting kernel problem. The solutions are evaluated, and the average R for the set is computed. The results for a set of problem sizes ranging from $N_f \simeq 75$ to 1000 are shown in tables 1.1 and 1.2, for the two real problems see table 1.3.

The results are quite impressive – the Potts algorithm has solved all problems, and with a very modest CPU time consumption, of which the major part goes into updating the P matrix. The sweep time scales like $(N_f^{\text{eff}})^3 \propto N_f^3$, with a small prefactor, due to the fast method used, eqs. (1.13, 1.14). This should be multiplied by the number of sweeps needed – empirically between 30 and 40, independently of problem size[6].

[6]The minor apparent deviation from the expected scaling in tables 1.1, 1.2 and 1.3 are due to an anomalous scaling of the Digital DXML library routines employed; the number of elementary operations does scale like N_f^3.

N_f	N_a	$\left\langle N_f^{\rm eff}\right\rangle$	$\left\langle N_a^{\rm eff}\right\rangle$	$\langle$R$\rangle$	$\langle$CPU time$\rangle$
75	5	23	8	0.0	0.0 sec
100	5	50	17	0.0	0.2 sec
150	10	55	17	0.0	0.3 sec
200	10	99	29	0.0	1.3 sec
225	15	84	26	0.0	0.7 sec
300	15	154	46	0.0	3.4 sec

Table 1.1: Average performance of the Potts algorithm for **LD** problems. The superscript "eff" refers to the kernel problem, subscript "f" and "a" refers to flight respectively airport. The averages are taken with 10 different problems for each N_f. The performance is measured as the difference between the waiting time in the Potts and the local solutions divided by the period. The CPU time refers to DEC Alpha 2000.

N_f	N_a	$\left\langle N_f^{\rm eff}\right\rangle$	$\left\langle N_a^{\rm eff}\right\rangle$	$\langle$R$\rangle$	$\langle$CPU time$\rangle$
600	40	280	64	0.0	19 sec
675	45	327	72	0.0	35 sec
700	35	370	83	0.0	56 sec
750	50	414	87	0.0	90 sec
800	40	441	91	0.0	164 sec
900	45	535	101	0.0	390 sec
1000	50	614	109	0.0	656 sec

Table 1.2: Average performance of the Potts algorithm for **SMD** problems. The averages are taken with 10 different problems for each N_f. Same notation as in table 1.1.

1.8 Summary

We have developed a mean field Potts approach for solving resource allocation problems with a non-trivial topology. The method is applied to airline crew scheduling problems resembling real-world situations.

A novel key feature is the handling of global entities, sensitive to the

N_f	N_a	$\left\langle N_f^{\text{eff}} \right\rangle$	$\left\langle N_a^{\text{eff}} \right\rangle$	$\langle \text{R} \rangle$	$\langle$CPU time$\rangle$	type
189	15	71	24	0.0	0.6 sec	LD
948	64	383	98	0.0	184 sec	SMD

Table 1.3: Average performance of the Potts algorithm for 10 runs on the two **real** problems. Same notation as in table 1.1.

dynamically changing "fuzzy" topology, by means of a propagator formalism. Another important ingredient is the problem size reduction achieved by airport fragmentation and flight clustering, narrowing down the solution space by removing much of the sub-optimal part.

High quality solutions are consistently found throughout a range of problem sizes without having to fine-tune the parameters, with a time consumption scaling as the cube of the problem size. The basic approach should be easy to adapt to other applications, like e.g. communication routing.

Bibliography

[1] J.J. Hopfield and D.W. Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* **52**, 141 (1985).

[2] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[3] L. Gislén, B. Söderberg and C. Peterson, "Teachers and Classes with Neural Networks", *International Journal of Neural Systems* **1**, 167 (1989).

[4] L. Gislen, B. Söderberg and C. Peterson, "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, 805 (1992).

[5] R. Durbin and D. Willshaw, "An Analog Approach to the Traveling Salesman Problem using an Elastic Net Method". *Nature* **326**, 689 (1987).

[6] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints - the Knapsack Problem", *Neural Computation* **5**, 331 (1993).

[7] D.W. Tank and J.J. Hopfield, "Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", *IEEE Transactions on Circuits and Systems* **CAS-33**, 533 (1986).

[8] See e.g. W.P. Press, B.P Flannery, S.A. Teukolsky and W.T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, Cambridge (1986).

Chapter 2

Airline Crew Scheduling Using Potts Mean Field Techniques

Paper II

LU TP 97-10
Revised

Airline Crew Scheduling Using Potts Mean Field Techniques

Martin Lagerholm[1], Carsten Peterson[2]
and Bo Söderberg[3]

Complex Systems Group,
Department of Theoretical Physics University of Lund,
Sölvegatan 14A, S-223 62 Lund, Sweden

Submitted to *European Journal of Operational Research*

[1]martin@thep.lu.se
[2]carsten@thep.lu.se
[3]bs@thep.lu.se

Abstract:

A novel method is presented and explored within the framework of Potts neural networks for solving optimization problems with a non-trivial topology, with the airline crew scheduling problem as a target application. The key ingredient to handle the topological complications is a propagator defined in terms of Potts neurons. The approach is tested on artificial problems generated with two real-world problems as templates. The results are compared against the properties of the corresponding unrestricted problems. The latter are subject to a detailed analysis in a companion paper [1]. Very good results are obtained for a variety of problem sizes. The computer time demand for the approach only grows like (number of flights)^3. A realistic problem typically is solved within minutes, partly due to a prior reduction of the problem size, based on an analysis of the local arrival/departure structure at the single airports.

To facilitate the reading for audiences not familiar with Potts neurons and mean field techniques, a brief review is given of recent advances in their application to resource allocation problems.

2.1 Introduction

Artificial Neural Networks (**ANN**) have over the last decade emerged as powerful tools for "intelligent" computing. Most attention has been paid to feed-forward architectures for pattern recognition and prediction problems. Conceptually, these approaches tie nicely into existing statistical and interpolation/extrapolation schemes. The application of feedback ANN methods to combinatorial optimization problems [2, 3, 4, 5] also looks very promising. In contrast to most search and heuristics methods, the ANN-based approach to optimization does not fully or partly explore the space of possible configurations; rather, the ANN "feels" its way through a continuous space of fuzzy configurations towards a good final solution. The intermediate fuzzy configurations have a natural probabilistic interpretation.

Typically, two basic steps are involved when using ANN to find good solutions to combinatorial optimization problems [6]: (1) map the problem onto a neural network (spin) system with a problem-specific energy function, and (2) minimize the energy by means of a deterministic process based on the iteration of mean field (MF) equations.

Initially, most applications concerned fairly artificial problems like the traveling salesman problem, various graph partition problems [2, 3] and knapsack problems [7, 8]. In refs. [9, 10], a more realistic problem (high school scheduling) was addressed. In all these applications, topological complication was not an issue, and could be dealt with in a straightforward way using "standard" ANN energy functions similar to those encountered in spin physics.

Recently, a formalism has been developed within the feedback ANN paradigm to handle applications with more complicated topologies, like airline crew scheduling and telecommunication routing problems [11, 12]. This paper deals with airline crew scheduling using the techniques briefly reported in ref. [11].

In airline crew scheduling, a given flight schedule is to be covered by a set of crew *rotations*, each consisting in a connected sequence of flights (*legs*), that begins and ends at a distinguished airport, the *home base*

(**HB**). The total crew time is to be minimized, subject to a number of restrictions on the rotations.

A commonly used approach to this problem proceeds in two steps. (1) First a large pool of feasible crew rotations that conform with the restrictions is generated (this is often referred to as column or matrix generation). (2) With such a set as a starting point, the problem is then reformulated as finding the best subset of rotations such that each flight is covered precisely once. This transforms the problem into a *set partitioning* problem (see e.g. [13] and references therein). Solutions to this "standard" problem are then found by approximate methods based on e.g. linear programming; more recently an exact branch-and-cut method has been used [13]. Even for moderate problem sizes, feasible rotations exist in astronomical numbers, and the pool has to be incomplete; this approach is therefore non-exhaustive.

Feedback ANN methods could be used to attack the resulting set partitioning problem. In fact, ANN methods have been successfully applied to the similar knapsack and set covering problems [7, 8, 14]. We will, however, follow a completely different pathway in approaching the airline crew scheduling problem: First, the full solution space is narrowed down using a reduction technique that removes a large part of the sub-optimal solutions. Then, an MF annealing approach based on a Potts neuron encoding is applied. A key feature here is the use of a recently developed propagator formalism [11] for handling topology, leg-counting, etc.

The method is explored on a set of synthetic problems, which are generated to resemble two real-world problems representing long and medium distance services. The algorithm performs well with respect to solution quality, with a computational requirement that at worst grows like N_f^3, where N_f is the number of flights.

The reduction technique employed, and the evaluation of the test problem results, rely heavily upon exploiting the properties of the solutions to the corresponding unconstrained problem, which decomposes into a local problem at each airport, and is solvable in polynomial time. A fairly extensive analysis of these properties is given in a companion article [1].

This paper is organized as follows: In Section 2 we define the problems under study, and Section 3 contains a discussion of the properties of the unrestricted local problems. Our method for initial reduction of the problem size is presented in Section 4. A generic brief review of the art of mapping resource allocation problems onto spin (neuron) systems, and a description of the MF annealing procedure, can be found in Section 5, and in Section 6 the Potts MF method for airline crew scheduling is presented. Section 7 contains performance measurements on a set of test problems, and finally in Section 8 we give a brief summary and outlook. Appendix A defines a toy problem that is used throughout the paper for illustrating the different techniques, while details on the Potts ANN algorithm and the problem generator can be found in Appendices B and C respectively.

2.2 Problem Definition

In a realistic airline crew scheduling problem one wants to minimize labour and other costs associated with a schedule of flights with specified times and airports of departure and arrival, subject to a number of safety and union constraints. Typically, a real-world flight schedule has a basic period of one week.

The problem considered in this work is somewhat stripped. We limit ourselves to minimizing the *total crew waiting-time*, subject to the constraints:

- The crews must follow connected flight sequences – *rotations* – starting and ending at the home-base.
- The number of flight legs in a rotation must not exceed a given upper bound.
- The total duration (flight-time + waiting-time) of a rotation is similarly bounded.

We believe that these are the crucial and difficult constraints; additional real-world constraints we have ignored do not constitute further challenges from an algorithmic point of view.

Throughout this paper, we will use a small toy problem, depicted in fig. 2.1, to illustrate our approach. The underlying flight data can be found in Appendix A.

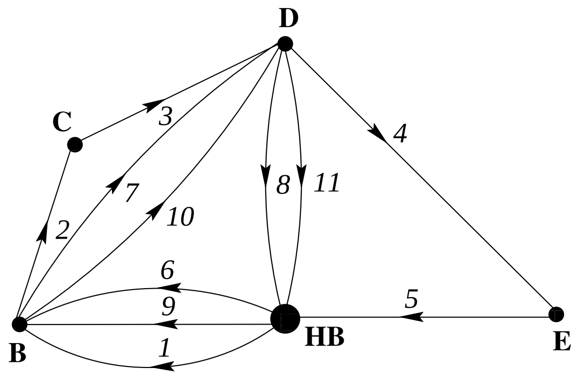

Figure 2.1: An illustration of the problem of table A.1.

Prior to developing our artificial neural network method, we will describe a technique to simplify the problem, based on an analysis of the local flight structure at each airport.

2.3 Properties of the Unrestricted Solutions

A solution to a given crew scheduling problem is specified by providing, at each airport (except HB), a one-to-one mapping between the arriving and departing flights. This implicitly defines the crew rotations.

It is the global constraints that make the crew scheduling problem a challenge. In the absence of these, there will be no interaction between the mappings at different airports; accordingly, the waiting-times can be minimized *independently* at each airport. This simplified problem will be referred to as the corresponding *unrestricted problem*; it is solvable in polynomial time. A detailed analysis of the statistical properties of such problems is presented in ref. [1]. Here we briefly describe the results from [1] needed for our preprocessing and analysis of the results.

Summing the resulting minimal waiting-times over the airports defines the minimal *unrestricted waiting-time*, denoted by $T^{\mathrm{unr}}_{\mathrm{wait}}$. This provides a lower bound to the minimal waiting-time for the full problem. Empirically, this bound is almost always saturated, i.e. among the minimal solutions to the unrestricted problem, a solution to the full problem can be found. This can be understood as follows.

At a single airport, the waiting-time for a given mapping is obtained by adding together the waiting-times for each arrival-departure pair (ij), given by

$$t_{ij}^{(\mathrm{w})} = \left(t_j^{(\mathrm{dep})} - t_i^{(\mathrm{arr})} \right) \bmod \text{period}. \tag{2.1}$$

Thus, the sum over pairs can only change by an integer number of periods. At a large airport, the minimum often is highly degenerate: For a random problem, the local ground-state degeneracy typically scales as $(N/2e)^N$ for an airport with $N \gg 1$ departures per period [1]. Consequently, the total number of minimal solutions to a complete unrestricted problem, defined as the product of the individual airport degeneracies, will be very large, and it is not inconceivable that a solution satisfying the constraints can be found among this set.

By insisting on ground-states, the state-space typically can be reduced by a factor of two for each flight. Part of this reduction is due to airports being split into smaller parts, which on the average gives a factor of two for each airport. This will be exploited in the next section, to reduce the size of a restricted problem. The unrestricted ground-states will also be used when gauging the performance of our Potts approach.

2.4 Reduction of Problem Size

By demanding a minimal waiting-time, the unrestricted local problem at each airport (excluding the home-base) typically can be further split up into independent *subproblems*, each containing a subset of the arrivals and an equally large subset of the departures. Some of these are trivial, forcing the crew of an arrival to continue to a particular departure.

Similarly, by demanding a solution with $T_{\mathrm{wait}}^{\mathrm{unr}}$ also for the constrained global problem, this can be reduced as follows:

- *Airport fragmentation*: Divide each airport into *effective airports* corresponding to the unrestricted local subproblems.
- *Flight clustering*: Join every forced sequence of flights into one effective *composite flight*, which will thus represent more than one leg and have a formal duration defined as the sum of the durations of its legs and the waiting-times between them.

Every problem will be preprocessed based on these two reduction methods, which will be explained in more detail below. In instances where no solution obeying the global constraints is found within the reduced solution space, one can attempt to solve the problem with no preprocessing. This was not necessary for any of the probed problems.

2.4.1 Airport Fragmentation

Inspecting the local arrival and departure times reveals which airports can be fragmented (for a full discussion, see ref. [1]). In the toy example of fig. 2.1, airports B and D can be split (see fig. 2.2). For airport B there

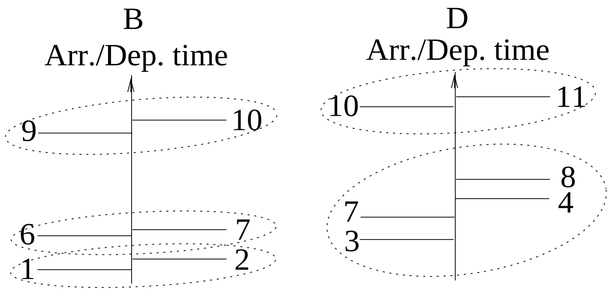

Figure 2.2: Arrival and departure times for airports B and D in the toy example. The dotted ellipses mark the fragments into which the airports can be divided.

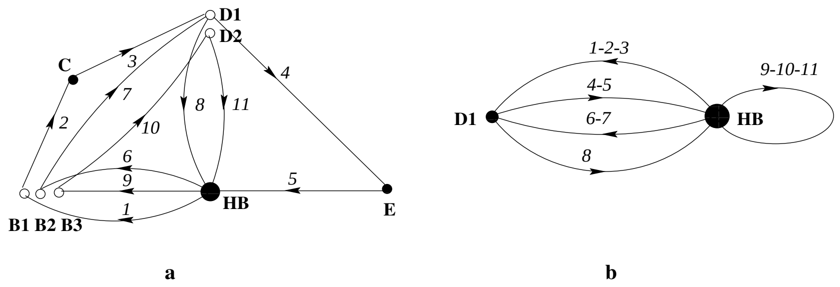

Figure 2.3: **(a)** The effective airports resulting from airport fragmentation for the toy-problem, and **(b)** the composite flights due to a subsequent flight clustering.

is only one possibility for connecting the flights without adding a period to the local waiting-time, yielding three effective airports (B1, B2 and B3). Similarly, airport D can be divided into two effective airports (D1 and D2). The structure that results from this fragmentation is shown in fig. 2.3a.

2.4.2 Flight Clustering

The airport fragmentation typically leads to several effective airports having only one arrival flight and one departure flight. Hence we can combine these into effective composite flights (flight clustering), with a formal duration obtained by adding together the flight duration times and the embedded waiting-times, and an intrinsic leg-count given by the number of proper flights included. The resulting structure for the toy problem is shown in fig. 2.3b and table A.2.

The *reduced problem* thus obtained differs from the original problem only in an essential reduction of the sub-optimal part of the solution space; the part with minimal waiting-time is unaffected. The resulting information gain, taken as the natural logarithm of the decrease in size of the solution space, empirically seems to scale approximately like $2\times$ (number of flights), and ranges from 100 to 2000 for the problems probed. This is considerably more than for a completely random, unstructured problem, where the gain is expected to scale like $\log 2\times$ (number of

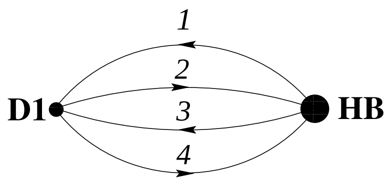

Figure 2.4: The kernel of the toy problem. The flights have been relabeled (see table A.2).

airports) [1].

2.4.3 The Kernel Problem

The reduced problem may in most cases be further separated into a set of independent *subproblems*, connected only via the home-base; these can be solved one by one. Some of the composite flights will formally arrive at the same effective airport they started from. This does not pose a problem; indeed, if the airport in question is the home-base, such a single flight constitutes a separate (trivial) subproblem, representing an entire forced rotation. Typically, one of the subproblems will be much larger than the rest, and will be referred to as the *kernel problem*, while the remaining subproblems will be essentially trivial.

In this way, our toy problem decomposes into two independent subproblems, one containing the single composite flight 9-10-11, the other containing the flights 1-2-3, 4-5, 6-7, and 8. The latter defines the kernel problem for our toy example. Relabeling the composite flights gives the structure shown in fig. 2.4. In the formalism below, we allow for the possibility that the problem to be solved has been reduced as described above, which means that flights may be composite. In what follows we limit ourselves to the kernel problem.

2.5 Optimization with Feedback Neural Networks

In this section we give a mini-review of how to map resource allocation problems onto feedback neural networks and the MF methodology for finding good solutions to such systems. Much of the formalism here originates from spin models in physics. Hence we will initially denote the basic degrees of freedom "spins". After discussing the MF approximation the term "neuron" will be used. We start out with a binary (Ising) system and then proceed to a multi-valued (Potts) system. The latter is the most relevant for the crew scheduling problem.

2.5.1 The Ising System

The Ising system is defined by the energy function

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j, \tag{2.2}$$

where the binary spins $\mathbf{s}_i$=±1 represent local magnetic properties of some material, and ω_{ij} how these spins couple to each other. Minimizing E in eq. (2.2) yields the spin configuration of the system.

Feedback networks for resource allocation problems with binary variables have a similar form. One such example is the graph bisection problem, where s_i encodes to what partition node i is assigned and w_{ij}=0,1 defines the problem in terms of whether i and j are connected or not. To enforce equal partition, $\sum s_i = 0$, eq. (2.2) needs to be augmented with a soft penalty term. One gets:

$$E = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j + \frac{\alpha}{2}\left(\sum_i s_i\right)^2, \tag{2.3}$$

equivalent to making the replacement $w_{ij} \to w_{ij} - \alpha$. The imbalance parameter α sets the relative strength between the cutsize and the balancing term. The next step is to find an efficient procedure for minimizing the energy in eqs. (2.2,2.3) aiming for the global minimum.

2.5.2 Ising Mean Field Equations

If one attempts to minimize the energy of eq. (2.3) according to a local optimization rule, the system will very likely end up in a local minimum close to the starting point, which is not desired. A better method is to use a stochastic algorithm that allows for uphill moves. One such method is *simulated annealing* (**SA**) [15], in which a sequence of configurations is generated, emulating the Boltzmann distribution

$$P[s] = \frac{1}{Z} e^{-E[s]/T}, \tag{2.4}$$

with neighbourhood search methods. In eq. (2.4), Z is the *partition function*,

$$Z = \sum_{[s]} e^{-E[s]/T}, \tag{2.5}$$

needed for normalization, and the width or *temperature* T represents the noise level of the system. For $T \to 0$ the Boltzmann distribution collapses into a delta function around the configuration minimizing E. By generating configurations at successively lower T (annealing) these are less likely to get stuck in local minima than if $T = 0$ from the start. Needless to say, such a procedure can be very CPU consuming.

The mean field (MF) approach aims at approximating the stochastic SA method with a deterministic process. This can be derived in two steps. First Z in eq. (2.5) is rewritten in terms of an integral over new continuous variables u_i and v_i. Then Z is approximated by the maximum value of its integrand.

To this end, introduce a new set of real-valued variables v_i, one for each spin, and set them equal to the spins with a Dirac delta-function. Then we can express the energy in terms of the new variables, and Z takes the form

$$\begin{aligned} Z &= \sum_{[s]} \int d[v] e^{-E[v]/T} \prod_i \delta(s_i - v_i) \\ &= \sum_{[s]} \int d[v] \int d[u] e^{-E[v]/T} \prod_i e^{u_i(s_i - v_i)}, \end{aligned} \tag{2.6}$$

where the delta-functions have been rewritten by introducing a new set of variables u_i. Then carry out the original sum over $[s]$ and write the product as a sum in the exponent:

$$Z \propto \int d[v] \int d[u] e^{-E[v]/T - \sum_i u_i v_i + \sum_i \log\cosh u_i} . \tag{2.7}$$

The original partition function is now rewritten entirely in terms of the new variables $[u, v]$, with an effective energy in the exponent. So far no approximation has been made. We next approximate Z in eq. (2.6) by the extremal value of the integrand obtained for

$$v_i = \tanh(u_i) = \tanh\left(-\frac{\partial E[v]}{\partial v_i}/T\right) . \tag{2.8}$$

The mean field variables (or *neurons*) v_i can be seen as approximations to the thermal averages $\langle s_i \rangle_T$ of the original binary spins. The MF equations (eq. (2.8)) are solved iteratively, either synchronously or asynchronously, under annealing in T. This defines a feedback ANN.

The dynamics of such an ANN typically exhibits a behaviour with two phases: When the temperature T is high, the sigmoid function, $\tanh(\cdot/T)$ in eq. (2.8), becomes very smooth, and the system relaxes into a trivial fixed point, $v_i^{(0)} = 0$. As the temperature is lowered a phase transition (bifurcation) occurs at $T = T_c$, where $v_i^{(0)}$ becomes unstable, and as $T \to 0$, fixed points $v_i^{(*)} = \pm 1$ emerge representing a specific decision made as to the solution to the optimization problems in question. The position of T_c depends upon w_{ij} and can be estimated by linearizing the sigmoid around $v_i^{(0)}$, i.e. linearizing eq. (2.8). Based on such an analysis, one can devise a reliable, parallelizable "black box" algorithm for solving problems of this kind.

Very good numerical results have been obtained for the graph bisection problem (see ref. [3] for references) for a wide range of problem sizes. The solutions are comparable in quality to those of the SA method, but the CPU time consumption is lower than any other known method of comparable performance. The approach of course becomes even more competitive with respect to time consumption if the intrinsic parallelism is exploited on dedicated hardware.

The MF approach differs fundamentally from many other heuristics, in that the evolution of the solutions starts outside the proper state space, and then gradually approaches the hypercube corners in solution space. This feature indicates a relation to interior point methods [16]. Indeed, as was pointed out in ref. [17], if the effective (or free) energy is convex, a variant of MF annealing can be obtained, which is equivalent to the interior point method [16].

2.5.3 The Potts System

For graph bisection and many other optimization problems, an encoding in terms of binary elementary variables is natural. However, there are many problems where this is not the case. In many cases it is more natural to replace the two-state Ising spins by multi-valued Potts spins, which have K possible values (states). For our purposes, the best representation of a Potts spin is in terms of a vector in the Euclidean space $\mathcal{E}_K$. Thus, denoting a spin variable by $\mathbf{s} = (s_1, s_2, \ldots, s_K)$, the j:th possible state is given by the j:th principal unit vector, defined by $s_j = 1$, $s_k = 0$ for $k \neq j$. These vectors point to the corners of a regular K-simplex. They are each normalized and fulfill the condition

$$\sum_k s_k = 1, \tag{2.9}$$

and they are mutually orthogonal,

2.5.4 Potts Mean Field Equations

The MF equations for a system of K-state Potts spins $\mathbf{s}_i = (s_{i1}, s_{i2}, \ldots, s_{iK})$ with an energy $E(\mathbf{s})$ are derived following the same path as in the Ising case – rewrite the partition function as an integral over $\mathbf{v}_i$ and $\mathbf{u}_i$ and approximate it with the maximum value of the integrand. One obtains

$$u_{ij} = -\frac{\partial E(\mathbf{v})}{\partial v_{ij}}/T, \tag{2.10}$$

$$v_{ij} = \frac{e^{u_{ij}}}{\sum_k e^{u_{ik}}}, \tag{2.11}$$

from which it follows that the MF *Potts neurons* $\mathbf{v}_i$, which approximate the thermal average of $\mathbf{s}_i$, satisfy

$$v_{ij} > 0\,,\ \sum_j v_{ij} = 1. \tag{2.12}$$

One can think of the neuron component v_{ij} as the *probability* for the i:th Potts spin to be in state j. For $K = 2$ one recovers the formalism of the Ising case in a slightly disguised form.

Supplying the Potts neurons with a dynamics based on iterating eqs. (2.10,2.11), yields a Potts ANN. Again one can typically analyze the linearized dynamics in order to estimate the critical temperature T_c. We refer the reader to ref. [3] for details.

It is often advantageous to replace the derivative in eq. (2.10) with the corresponding difference,

$$u_{ij} = -\left(E|_{v_{ij}=1} - E|_{v_{ij}=0}\right)/T, \tag{2.13}$$

which will be used in the airline crew problem below.

2.6 Potts Neural Approach to the Crew Scheduling Problem

2.6.1 Encoding

We are now ready to encode the airline crew problem in terms of Potts spins. A naive way to do this would be to mimic what was done in the teachers-and-classes problem in refs. [9, 10], where each event (lecture) was mapped onto a resource unit (lecture-room + time-slot). This would require a Potts spin for each flight to handle the mapping onto crews.

Since the problem consists in linking together sequences of (composite) flights into rotations, it appears more natural to choose an encoding where each flight i is mapped, via a Potts spin, onto the flight j to follow it in the rotation:

$$s_{ij} = \begin{cases} 1 & \text{if flight } i \text{ precedes flight } j \text{ in a rotation,} \\ 0 & \text{otherwise,} \end{cases}$$

where it is understood that j be restricted to depart from the (effective) airport where i arrives. In order to ensure that proper rotations are formed, each flight has to be mapped onto precisely one other flight. This restriction is inherent in the Potts spin formulation, which is defined to have precisely one component "on", as is evident from eq. (2.9).

To start or terminate a rotation, we introduce dummy flights a and b of zero duration and intrinsic leg count, available only at the home-base, representing the start/end of a rotation – at the home-base, a is formally mapped onto every departure, and every arrival is mapped onto b.

We illustrate the Potts encoding by one particular solution to the toy kernel problem of fig. 2.4, where flight 1 is connected to flight 2, and flight 3 to flight 4. In eq. (2.14) the "border" entries of $\mathbf{s}$ corresponding to the dummy flights a $(i,j=0)$ and b $(i,j=N+1)$ are marked in bold face.

$$\mathbf{s} = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 0 & 1 & 0 & 0 & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 0 & 0 & \mathbf{1} \\ \mathbf{0} & 0 & 0 & 0 & 1 & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 0 & 0 & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.14}$$

Global topological properties, leg-counts and durations of rotations, etc., cannot be described in a simple way by polynomial functions of the spins. Instead, they are conveniently handled by means of a *propagator* matrix $\mathbf{P}$, defined in terms of the Potts spin matrix $\mathbf{s}$ by

$$\begin{aligned} P_{ij} &= \left((\mathbf{1}-\mathbf{s})^{-1}\right)_{ij} = \delta_{ij} + s_{ij} + \sum_k s_{ik}s_{kj} \\ &+ \sum_{kl} s_{ik}s_{kl}s_{lj} + \sum_{klm} s_{ik}s_{kl}s_{lm}s_{mj} + \ldots \end{aligned} \tag{2.15}$$

A pictorial expansion of the propagator is shown in fig. 2.5. The interpretation is obvious: P_{ij} counts the number of connecting paths

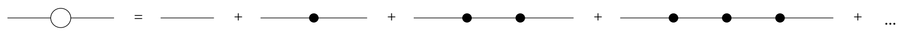

Figure 2.5: Expansion of the propagator P_{ij} (◯) in terms of s_{ij}. A line represents a flight, and (•) a landing.

from flight i to j. The P-matrix corresponding to the toy problem solution of eq. (2.14) is given by

$$\mathbf{P} = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{2} \\ \mathbf{0} & 1 & 1 & 0 & 0 & \mathbf{1} \\ \mathbf{0} & 0 & 1 & 0 & 0 & \mathbf{1} \\ \mathbf{0} & 0 & 0 & 1 & 1 & \mathbf{1} \\ \mathbf{0} & 0 & 0 & 0 & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}, \tag{2.16}$$

where one finds two paths from a to b ($P_{ab} = 2$), as there should be – one via 1-2 and one via 3-4.

Similarly, an element of the matrix square of $\mathbf{P}$,

$$\left(P^2\right)_{ij} \equiv \sum_k P_{ik} P_{kj} = \delta_{ij} + 2 s_{ij} + 3 \sum_k s_{ik} s_{kj} + \ldots, \tag{2.17}$$

counts the total number of composite legs in the connecting paths between i and j, while the number of proper legs is given by

$$\begin{aligned} \tilde{L}_{ij} &\equiv \sum_k P_{ik} L_k P_{kj} = \delta_{ij} L_i + s_{ij} \left(L_i + L_j\right) \\ &+ \sum_k s_{ik} s_{kj} \left(L_i + L_k + L_j\right) + \ldots, \end{aligned} \tag{2.18}$$

where L_k is the intrinsic number of single legs in the composite flight k. For the toy model solution, we get

$$\mathbf{P}^2 = \begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{2} & \mathbf{3} & \mathbf{8} \\ \mathbf{0} & 1 & 2 & 0 & 0 & \mathbf{3} \\ \mathbf{0} & 0 & 1 & 0 & 0 & \mathbf{2} \\ \mathbf{0} & 0 & 0 & 1 & 2 & \mathbf{3} \\ \mathbf{0} & 0 & 0 & 0 & 1 & \mathbf{2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix} \tag{2.19}$$

and

$$\tilde{\mathbf{L}} = \begin{pmatrix} \mathbf{0} & \mathbf{3} & \mathbf{5} & \mathbf{2} & \mathbf{3} & \mathbf{8} \\ \mathbf{0} & 3 & 5 & 0 & 0 & \mathbf{5} \\ \mathbf{0} & 0 & 2 & 0 & 0 & \mathbf{2} \\ \mathbf{0} & 0 & 0 & 2 & 3 & \mathbf{3} \\ \mathbf{0} & 0 & 0 & 0 & 1 & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.20}$$

based on the intrinsic leg counts 3, 2, 2, and 1, for the flights 1, 2, 3, and 4, respectively. The path via 1-2 has 5 legs, and the one via 3-4 has 3 legs, making a total leg count of 8 for all paths from a to b, as can be read off from the upper right corner ($\tilde{L}_{ab} = 8$).

The *average leg count* of the connecting paths is then given by

$$L_{ij} \equiv \frac{\tilde{L}_{ij}}{P_{ij}}, \tag{2.21}$$

and for the *average duration* (flight + waiting-time) of the paths from i to j one has

$$T_{ij} \equiv \frac{\sum_k P_{ik} t_k^{(\mathrm{f})} P_{kj} + \sum_{kl} P_{ik} t_{kl}^{(\mathrm{w})} s_{kl} P_{lj}}{P_{ij}}, \tag{2.22}$$

where $t_i^{(\mathrm{f})}$ denotes the duration of the composite flight i, including the embedded waiting-time. The averaging is accomplished by the division with P_{ij}. In principle, this could lead to undefined expressions in cases with $P_{ij} = 0$. This will be no problem, since we will only be interested in cases either with $i = a$, probing the path from a to j, i.e. the part up to j of the rotation containing j, or $j = b$, probing the path from i to b, i.e. the part after and including i of the rotation to which i belongs.

Furthermore, any improper loops (such as obtained e.g. if two flights are mapped onto each other) will make $\mathbf{P}$ singular – for a proper set of rotations, $\det \mathbf{P} = 1$.

2.6.2 Mean Field Dynamics

In the MF formalism the basic dynamical variables are $\mathbf{v}$ rather than $\mathbf{s}$; correspondingly, we will use a probabilistic propagator $\mathbf{P}$, defined as the matrix inverse of $\mathbf{1}-\mathbf{v}$, in analogy with eq. (2.15), but with $\mathbf{s}$ replaced by $\mathbf{v}$. The clearcut structure seen in the toy-model matrices in eqs. (2.14,2.16), will only emerge as $T \to 0$.

Rather than finding a suitable energy function in terms of the matrices $\mathbf{v}$ and $\mathbf{P}$, we have chosen a more pragmatic approach by directly writing down the *local fields* u_{ij}, bypassing eq. (2.10). The corresponding mean fields v_{ij} are obtained from the MF equations (eq. (2.11)); they have an obvious interpretation of probabilities (for flight i to be followed by j).

In the MF equations (eq. (2.11)) v_{ij} will be updated for one flight i at a time, by first zeroing the i:th row of $\mathbf{v}$ (and updating $\mathbf{P}$ correspondingly), and then computing the relevant local fields u_{ij} entering eq. (2.11) as

$$\begin{aligned} u_{ij} &= -\frac{c_1}{T} t_{ij}^{(\mathrm{w})} - \frac{c_2}{T} \sum_k v_{kj} - \frac{c_3}{T} \log\left(\frac{1}{1-P_{ji}}\right) \\ &- \frac{c_4}{T} \Psi\left(T_{\mathrm{rot}}^{(\mathrm{ij})} - T_{\mathrm{rot}}^{\max}\right) - \frac{c_5}{T} \Psi\left(L_{\mathrm{rot}}^{(\mathrm{ij})} - L_{\mathrm{rot}}^{\max}\right), \end{aligned} \tag{2.23}$$

where j is restricted to be a possible continuation flight to i. It is difficult, and not necessary from the viewpoint of algorithmic performance, to find energy functions corresponding to the fourth and fifth terms in eq. (2.23). In contrast, the first and second terms are straightforward in this respect, and the third term originates from an energy term $\sim \log\det \mathbf{P}$. The five different terms in eq. (2.23) serve the following purposes:

1. Cost term: The local waiting-time $t_{ij}^{(\mathrm{w})}$ between flight i and j.
2. Penalizes solutions where two flights point to the same next flight.
3. Suppresses improper loops. $P_{ji} \to 1$ if a path $j \to i$ exists, i.e. if a loop is formed if i connects with j. The penalty approaches ∞ when $P_{ji} \to 1$.
4. Prohibits violation of the bound $T_{\mathrm{rot}}^{\max}$ on total rotation time, where $T_{\mathrm{rot}}^{(\mathrm{ij})}$ stands for the duration of the resulting rotation if i where to be mapped onto j.
5. Prohibits violation of the bound $L_{\mathrm{rot}}^{\max}$ on total number of legs, where $L_{\mathrm{rot}}^{(\mathrm{ij})}$ is the resulting number of legs in the rotation if i where to be mapped onto j.

In eq. (2.23), the rotation time $T_{\rm rot}^{\rm (ij)}$ and the leg count $L_{\rm rot}^{\rm (ij)}$ are given as

$$
\begin{aligned}
T_{\rm rot}^{\rm (ij)} &= T_{ai} + t_{ij}^{\rm (w)} + T_{jb}, & (2.24)\\
L_{\rm rot}^{\rm (ij)} &= L_{ai} + L_{jb}, & (2.25)
\end{aligned}
$$

in terms of eqs. (2.22,2.21).

The penalty function Ψ, used to enforce the inequality constraints [7], is defined by $\Psi(x) = x\Theta(x)$ where Θ is the Heaviside step function. It turns out, as will be discussed below, that the performance of the algorithm is fairly insensitive to the choice of the relative strengths c_i occurring in eq. (2.23).

After an initial computation of the propagator $\mathbf{P}$ from scratch, it is subsequently updated according to the Sherman-Morrison algorithm for incremental matrix inversion [19]. An update of the i:th row of $\mathbf{v}$, $v_{ij} \to v_{ij} + \delta_j$, generates precisely the following change in the propagator $\mathbf{P}$:

$$
P_{kl} \to P_{kl} + \frac{P_{ki}\, z_l}{1 - z_i}, \tag{2.26}
$$

with

$$
z_l = \sum_j \delta_j\, P_{jl}. \tag{2.27}
$$

Inverting the matrix from scratch would take $O(N^3)$ operations, while the (exact) scheme devised above only requires $O(N^2)$ per row.

In principle, a proper value for an initial temperature can be estimated from linearizing the dynamics of the MF equations. The neurons are initialized close to the trivial fixed point. A common annealing schedule for the updating, based on iterating the MF eqs. (2.11,2.26), is to decrease T by a fixed factor per iteration.

As the temperature goes to zero, a solution crystallizes in a winner-takes-all dynamics: for each flight i, the largest u_{ij} determines the continuation flight j to be chosen.

Implementation details of the algorithm can be found in Appendix B.

Parallelizing the Algorithm

One obstacle, if one wants to parallelize the algorithm, is that the scheme above, eqs. (2.26,2.27), for updating $\mathbf{P}$ is non-local, in that all matrix elements of $\mathbf{P}$ are updated due to a change in a single neuron $\mathbf{v}_i$. An alternative method using only local information on $\mathbf{v}$ and $\mathbf{P}$ is to update row i of $\mathbf{P}$ according to

$$P_{im} \to \delta_{im} + \sum_j v_{ij} P_{jm}, \tag{2.28}$$

in connection with updating the corresponding row of $\mathbf{v}$. If each flight keeps track of its own row of $\mathbf{P}$, all information needed can be obtained from the possible continuation flights j ("neighbours") to flight i. This scheme gives convergence towards the exact inverse; a similar method has been successfully used in the context of communication routing [12].

The handling of the global time and leg constraints of a rotation could be tackled in a similar manner, with each flight keeping track of the time and number of legs used both from a to itself and from itself to b, where a and b are the dummy flights starting and terminating a rotation. The information needed to calculate $T_{\text{rot}}^{(\text{ij})}$ and $L_{\text{rot}}^{(\text{ij})}$ then is local to i and its "neighbours" j.

2.7 Test Problems

In choosing test problems our aim has been to maintain a reasonable degree of realism, while avoiding unnecessary complication and at the same time not limiting ourselves to a few real-world problems, where one can always tune parameters and procedures to get a good performance. In order to accomplish this we have analyzed two typical real-world *template* problems obtained from a major airline: one consisting of long distance (LD), the other of short/medium distance (SMD) flights. As can be seen from fig. 2.6, LD flight time distributions are centered around long times, with a small hump for shorter times representing local continuations of long flights. The SMD flight times have a more compact distribution.

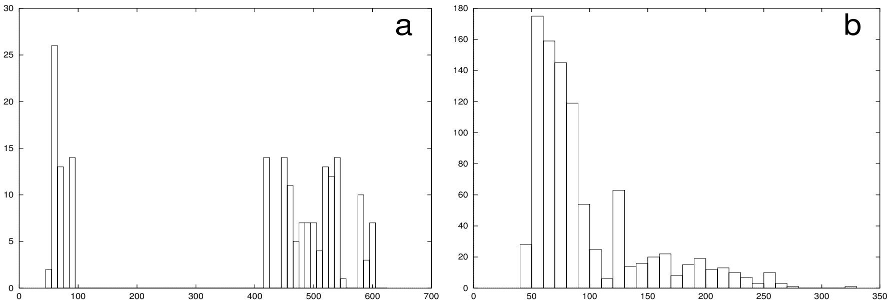

Figure 2.6: Flight time distributions in minutes for **(a)** LD and **(b)** SMD template problems.

For each template we have made a distinct problem generator producing random problems resembling the template. For algorithm details see Appendix C.

Due to the excessive time consumption of the available exact methods, the performance of the Potts approach cannot be tested against these – except for in this context quite small problems, for which the Potts solution quality matches that of an exact algorithm. For artificial problems of more realistic size we circumvent this obstacle in the following way: since problems are generated by producing a legal set of rotations, we add in the generator a final check that the implied solution yields $T_{\mathrm{wait}}^{\mathrm{unr}}$; if not, a new problem is generated. Theoretically, this might introduce a bias in the problem ensemble; empirically, however, no problems have had to be redone. Also the two template problems turn out to be solvable at $T_{\mathrm{wait}}^{\mathrm{unr}}$.

Each problem then is reduced as described above (using a negligible amount of computer time), and the kernel problem is stored as a list of flights, with all traces of the generating rotations removed.

2.8 Results

We have tested the performance of the Potts MF approach for both LD and SMD kernel problems of varying sizes.

c_1	c_2	c_3	c_4	c_5
period^{-1}	1	1	$\langle T^{\mathrm{rot}}\rangle^{-1}$	$\langle L^{\mathrm{rot}}\rangle^{-1}$

Table 2.1: The coefficients used in eq. (2.23). $\langle T^{\mathrm{rot}}\rangle$ is the average duration per rotation (based on $T^{\mathrm{unr}}_{\mathrm{wait}}$), and $\langle L^{\mathrm{rot}}\rangle$ the average leg count, both of which can be computed beforehand.

The values used for the coefficients c_i in eq. (2.23) are displayed in table 2.1. One should stress that these parameter settings have been used for the entire range of problem sizes probed. For the LD problems the bounds on a rotation are chosen as

$$T^{\mathrm{max}}_{\mathrm{rot}} = 10000, \quad (2.29)$$
$$L^{\mathrm{max}}_{\mathrm{rot}} = 15, \quad (2.30)$$

and to

$$T^{\mathrm{max}}_{\mathrm{rot}} = 6000, \quad (2.31)$$
$$L^{\mathrm{max}}_{\mathrm{rot}} = 25, \quad (2.32)$$

for the SMD problems.

A typical evolution of the individual neuron components v_{ij} is shown in fig. 2.7. In fig. 2.8 the evolution of the number of legs of all the rotations (defined by L_{ib} where i is a departure flight from HB) for two different values of the bound $L^{\mathrm{max}}_{\mathrm{rot}}$.

When evaluating a solution obtained with the Potts approach, a check is done as to whether it is legal (if not, a simple post-processor tries to restores legality – this is only occasionally needed), then the solution quality is probed by measuring the excess waiting-time R,

$$R = \frac{T^{\mathrm{wait}}_{\mathrm{Potts}} - T^{\mathrm{unr}}_{\mathrm{wait}}}{\mathrm{period}}, \quad (2.33)$$

which is a non-negative integer for a legal solution.

For a given problem size, as given by the desired number of airports N_a and flights N_f, a set of 10 distinct problems is generated. Each

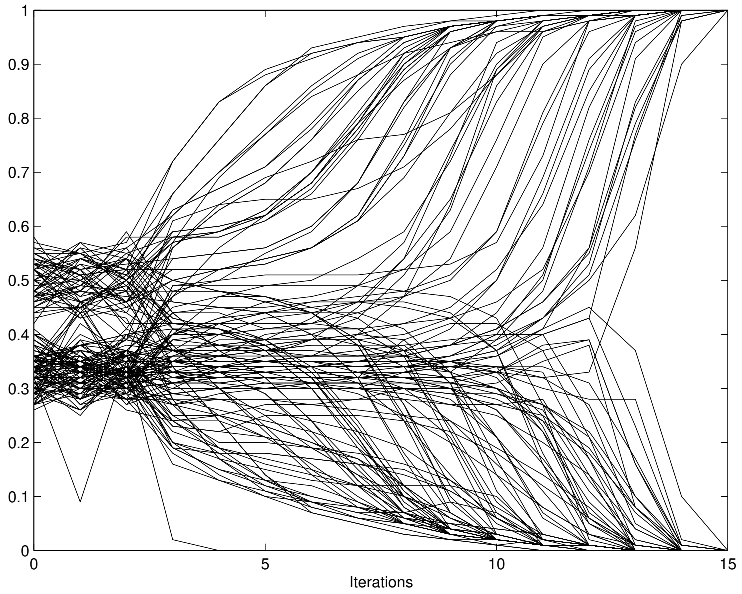

Figure 2.7: Evolution of neuron components v_{ij} as the temperature is lowered for the template LD problem.

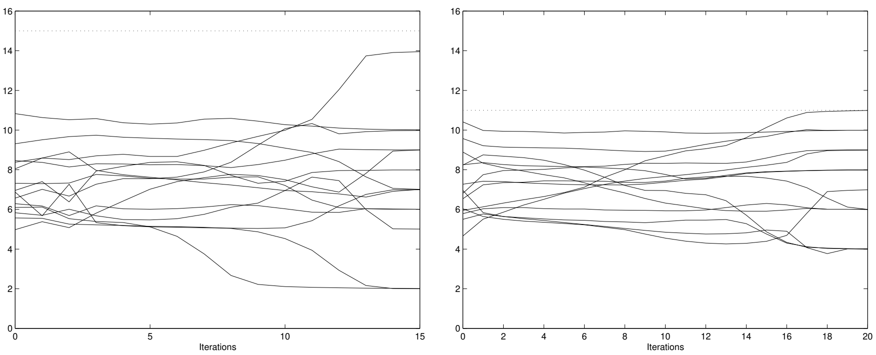

Figure 2.8: Evolution of L_{ib} for all departure flights from the home base, i, for the template LD problem. The dotted lines denote $L_{\text{rot}}^{\max}$.

problem is subsequently reduced, and the Potts algorithm is applied to the resulting kernel problem. The solutions are evaluated, and the

N_f	N_a	$\left\langle N_f^{\text{eff}} \right\rangle$	$\left\langle N_a^{\text{eff}} \right\rangle$	$\langle \text{R} \rangle$	$\langle \text{CPU time} \rangle$
75	5	23	8	0.0	0.0 sec
100	5	44	13	0.0	0.2 sec
150	10	46	14	0.0	0.1 sec
200	10	84	24	0.0	0.7 sec
225	15	74	22	0.0	0.4 sec
300	15	132	38	0.0	1.5 sec

Table 2.2: Average performance of the Potts algorithm for **LD** problems. The superscript "eff" refers to the kernel problem, subscripts "f" and "a" refer to respectively flight and airport. The averages are taken with 10 different problems for each N_f. R is the excess waiting-time, and the CPU time refers to DEC Alpha 2000.

average R for the set is computed. The results for a set of problem sizes ranging from $N_f \simeq 75$ to 1000 are shown in tables 2.2 and 2.3; for the two template problems see table 2.4.

N_f	N_a	$\left\langle N_f^{\text{eff}} \right\rangle$	$\left\langle N_a^{\text{eff}} \right\rangle$	$\langle \text{R} \rangle$	$\langle \text{CPU time} \rangle$
600	40	280	64	0.0	19 sec
675	45	327	72	0.0	35 sec
700	35	370	83	0.0	56 sec
750	50	414	87	0.0	90 sec
800	40	441	91	0.0	164 sec
900	45	535	101	0.0	390 sec
1000	50	614	109	0.0	656 sec

Table 2.3: Average performance of the Potts algorithm for **SMD** problems. The averages are taken with 10 different problems for each size. Same notation as in table 2.2.

The results are quite impressive – the Potts algorithm has solved all problems, and with a very modest CPU time consumption, of which the major part is used in updating the P matrix, using the fast method of eqs. (2.26, 2.27). The iteration time scales like $(N_f^{\text{eff}})^3 \propto N_f^3$ with a

N_f	N_a	$\left\langle N_f^{\text{eff}} \right\rangle$	$\left\langle N_a^{\text{eff}} \right\rangle$	$\langle$R$\rangle$	$\langle$CPU time$\rangle$	type
189	15	71	24	0.0	0.6 sec	LD
948	64	383	98	0.0	184 sec	SMD

Table 2.4: Average performance of the Potts algorithm for 10 runs on the two template problems. Same notation as in table 2.2.

small prefactor. This should be multiplied by the number of iterations needed – empirically between 20 and 40, independently of problem size[4].

2.9 Summary

We have developed a mean field Potts approach for finding good solutions to airline crew scheduling problems resembling real-world situations.

A key feature is the handling of global entities, sensitive to the dynamically changing "fuzzy" topology, by means of a propagator formalism. This is a novel ingredient in ANN-based approaches to resource allocation problems. Another important ingredient is the problem size reduction achieved by airport fragmentation and flight clustering, narrowing down the solution space by removing much of the sub-optimal part. This is done by exploiting the local properties of the corresponding unrestricted problems [1].

High quality solutions are consistently found throughout a range of problem sizes without having to fine-tune the parameters, with a time consumption scaling as the cube of the problem size. The performance of the Potts algorithm is probed by comparing to the unrestricted optimal solutions.

At first sight, the Potts algorithm appears difficult to implement in a

[4]The minor apparent deviation from the expected scaling in tables 2.2, 2.3 and 2.4 are due to an anomalous scaling of the Digital DXML library routines employed; the number of elementary operations does scale like N_f^3.

parallel way with its global quantities. A concurrent implementations can be facilitated, however, by localizing all information.

The basic approach presented here is easy to adapt to other applications, in particular in communication routing [12].

Acknowledgements:

We thank Richard Blankenbecler for valuable suggestions on the manuscript. This work was in part supported by the Swedish Natural Science Research Council and the Swedish Board for Industrial and Technical Development.

Appendix A

A Toy Example

In this Appendix we define and analyze a small toy example with five airports, that is used throughout this paper to illustrate the various steps. When exploring the performance of the algorithm, much larger problems are of course involved (see Section 7).

The toy problem is specified in table A.1, where a period of 10080 is assumed. An illustration is shown in fig. 2.1.

Flight	Dep. airport	Arr. airport	Dep. time	Arr. time
1	HB	B	0	500
2	B	C	1000	1300
3	C	D	1500	1850
4	D	E	4300	4870
5	E	HB	5100	5500
6	HB	B	1500	2000
7	B	D	2200	2800
8	D	HB	3500	4100
9	HB	B	6000	6500
10	B	D	7000	7500
11	D	HB	8000	8250

Table A.1: Toy problem specification.

Fragmentation and clustering with flight duration times and the number of legs added together gives the structures shown in fig. 2.3b and table A.2. The corresponding kernel problem is shown in fig. 2.4 with the effective flights relabeled according to table A.2. The total flight-time

Comp. flight	Leg-count	Legs	Dep. airport	Arr. airport	Dep. time	Arr. time
1	3	1-2-3	HB	D1	0	1850
2	2	4-5	D1	HB	4300	5500
3	2	6-7	HB	D1	1500	2800
4	1	8	D1	HB	3500	4100
5	3	9-10-11	HB	HB	6000	8250

Table A.2: Toy model description after fragmentation and clustering. The first column gives the composite flight label.

is 5070, and without restrictions, there are two solutions with minimum waiting-time, 5280. One consists in the rotations 1-2, 3-4, and 5 in terms of composite flights, i.e. 1-2-3-4-5, 6-7-8, and 9-10-11 in terms of the original flights. The corresponding rotation times are 5500, 2600 and 2250, respectively. The other has the rotations 1-4, 2-3, and 5 in terms of composite flights, i.e. 1-2-3-8, 6-7-4-5, and 9-10-11 in terms of proper flights, with the respective rotation times 4100, 4000, and 2250.

Appendix B

The Potts Algorithm

Initialization

The initial temperature T_0 is assigned a tentative value of 1.0. If the averaged squared change of the neurons,

$$(\Delta v)^2 = \frac{1}{\tilde{N}_f} \sum_{ij} (\Delta v_{ij})^2 = \frac{1}{\tilde{N}_f} \sum_{ij} (v_{ij}(t+1) - v_{ij}(t))^2, \tag{B.1}$$

is larger than 0.2 after the first iteration, then the system is reinitialized with $T_0 \to 2T_0$. If, on the other hand, it is smaller than 0.01 the system is reinitialized with $T_0 \to T_0/2$. In eq. (B.1) $\tilde{N}_f$ is the number of flights minus those departing from HB,

Each neuron $\mathbf{v}_i$ is initialized by assigning random values to its components v_{ij} in the interval $0.8/K$ to $1.2/K$, where K is the number of components of the Potts neuron. The neuron is then normalized by dividing each component by the component sum.

Subsequently, P_{ij}, T_{ij} and L_{ij} are initialized consistently with the neuron values.

The following iteration is repeated, until one of the termination criteria (see below) is fulfilled:

Iteration

- For each airport (in random order) do:
  1. For each arrival flight i, do:
     (a) Update $\mathbf{v}_i$ (eqs. (2.11,2.23))
     (b) Update $\mathbf{P}$ (eq. (2.26)
  2. Correct the neuron matrix by doing the following N_{norm} times:
     (a) Normalize the columns of $\mathbf{v}$, corresponding to local departures.
     (b) Normalize the rows of $\mathbf{v}$, corresponding to local arrivals.
- Decrease the temperature: $T = kT$.

We have consistently used $k = 0.9$ and $N_{\text{norm}} = 2$.

Termination criteria

The updating process is terminated if

$$\begin{aligned} &1/\tilde{N}_f \sum_{ij} v_{ij}^2 > 0.99 \;\text{ and }\; \max_{ij}(\Delta v_{ij})^2 < 0.01 \\ &\text{and }\; \min_{ij} v_{ij}^2 > 0.8, \end{aligned} \tag{B.2}$$

or

$$\begin{aligned} &1/\tilde{N}_f \sum_{ij} v_{ij}^2 > 0.8 \;\text{ and }\; (\Delta v)^2 > 0.000001 \\ &\text{and }\; \max_{ij}(\Delta v_{ij})^2 < 0.01 \;\text{ and }\; \min_{ij} v_{ij}^2 > 0.8, \end{aligned} \tag{B.3}$$

or if the number of iterations exceeds 100.

Postprocessing

First, the final state of each neuron $\mathbf{v}_i$ is analyzed with respect to its implied choice, defined by its largest component. A check is done as to whether a proper rotation structure results. If this is not the case (which never happened for the problems studied here), one may e.g. rerun the algorithm with modified parameters.

Then, each rotation is checked for legality: If the rotation time or leg count exceeds the respective bound, a simple algorithm is employed to attempt to restore legality, by swapping flights between rotations. For the few cases (~ 5 %) where such a correction was needed for the problems studied here, this procedure always sufficed.

Appendix C

The Problem Generator

We have made two distinct problem generators, tuned to generate problems statistically resembling the LD or SMD templates. In both generators, a random problem with a specified number of airports and flights is generated as follows.

First, the flight-times between airports are chosen randomly from a distribution, based on the relevant template problem. Then, a flight schedule is built up in the form of legal rotations starting and ending at the home-base. The waiting-times between consecutively flights are chosen in a random fashion in the neighbourhood of $T^{\text{unr}}_{\text{wait}}/N_f$ for the corresponding template problem.

For a specified number of airports N_a and flights N_f, the key steps take the following form.

Generator steps

- The airports are assigned distinct probabilities, designed to match the traffic distribution for the airports in the relevant template problem.
- For each pair of airports, a distance (flight-time) is drawn from a predefined distribution.
- While the number of generated flights is less than N_f:
  Start a new rotation from HB, then for each leg do:
  1. Choose its destination:
     - If the number of legs is less than L_0, draw the destination from a predefined distribution, where HB is chosen with a probability $\mathcal{P}_{\text{HB}}$. [a] [b]
     - Else force the destination to be HB, and begin a new rotation.
  2. Pick the waiting-time from the predefined distribution.
  3. Set the flight time according to the distance table, with some random deviation.
- If any rotation time exceeds the limit, or if the solution does not end up at the unrestricted minimal waiting-time, generate a new problem.

[a] Care is taken that the very last leg goes to HB.

[b] If more than half of the flights are generated and some airports still are not visited, then if the destination is not HB, change to an unvisited airport. Only one airport per rotation is allowed to be chosen in this way.

The probability $\mathcal{P}_{\text{HB}}$ for choosing HB as a destination is for both problem types chosen to 0.25, except for the first leg, giving on the average 5 legs per rotation. For LD-problems, the maximum legcount L_0 is set to 15, while for the SMD problems it is set to 25.

Bibliography

[1] M. Lagerholm, C. Peterson and B. Söderberg, "Statistical Properties of Unrestricted Crew Scheduling Problems", *LU TP 97-11* (submitted to *European Journal of Operational Research*).

[2] J.J. Hopfield and D.W. Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* **52**, 141 (1985).

[3] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[4] R. Durbin and D. Willshaw, "An Analog Approach to the Traveling Salesman Problem using an Elastic Net Method". *Nature* **326**, 689 (1987).

[5] C. Peterson, "Parallel Distributed Approaches to Combinatorial Optimization Problems – Benchmark Studies on TSP", *Neural Computation* **2**, 261 (1990).

[6] C. Peterson and B. Söderberg, "Artificial Neural Networks and Combinatorial Optimization Problems", in *Local Search in Combinatorial Optimization*,
eds. E.H.L. Aarts and J.K. Lenstra. New York 1997: John Wiley & Sons.

[7] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints - the Knapsack Problem", *Neural Computation* **5**, 331 (1993).

[8] M. Ohlsson and H.Pi, "A study of the Mean Field Approach to Knapsack Problems", *Neural Networks* **10**, 263 (1997).

[9] L. Gislén, B. Söderberg and C. Peterson, "Teachers and Classes with Neural Networks", *International Journal of Neural Systems* **1**, 167 (1989).

[10] L. Gislén, B. Söderberg and C. Peterson, "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, 805 (1992).

[11] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling with Potts Neurons", *LU TP 96-6* (to appear in *Neural Computation*).

[12] J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg, "A Potts Neuron Approach to Communication Routing" *LU TP 97-2* (submitted to *Neural Computation*).

[13] K.L. Hoffman and M. Padberg, "Solving Airline Crew Scheduling Problems by Branch-and-Cut", *Management Science* **39**, 657 (1993).

[14] M. Ohlsson, C. Peterson and B. Söderberg, "A Mean Field Annealing Algorithm for the Set Covering Problem", in preparation.

[15] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science* **220**, 671 (1983).

[16] N. Karmarkar, "A New Polynomial-time Algorithm for Linear Programming", *Combinatorica* **4**, 373 (1984).

[17] A.L. Yuille, "Statistical Physics Algorithms that Converge", *Neural Computation* **6**, 341 (1994).

[18] L. Faybusovich, "Interior Point Methods and Entropy", *Proc. of the IEEE Conference on Decision and Control* **3**, 2094 (1991).

[19] See e.g. W.P. Press, B.P Flannery, S.A. Teukolsky and W.T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, Cambridge (1986).

Chapter 3

Statistical Properties of Unrestricted Crew Scheduling Problems

Paper III

LU TP 97-11
Revised

Statistical Properties of Unrestricted Crew Scheduling Problems

Martin Lagerholm[1], Carsten Peterson[2]
and Bo Söderberg[3]

Complex Systems Group,
Department of Theoretical Physics University of Lund,
Sölvegatan 14A, S-223 62 Lund, Sweden

Submitted to *European Journal of Operational Research*

[1]martin@thep.lu.se
[2]carsten@thep.lu.se
[3]bs@thep.lu.se

Abstract:

A statistical analysis is performed for a random unrestricted local crew scheduling problem, expressed in terms of pairing arrivals with departures. The analysis is aimed at understanding the structure of similar problems with global restrictions, and estimating their difficulty. The methods developed are of a general nature and can be of use in other problems with a similar structure. For large random problems, the ground-state energy scales like $\sqrt{N}$ and the average excitation like N, where N is the number of arrivals/departures. The average ground-state degeneracy is such that the probability of hitting an optimal pairing by chance scales like $2N2^{-N}$ for large N. By insisting on the local ground-state energy for a restricted problem, airports can be split into smaller parts, and the state space reduced by typically a factor $\sim 2^{N_a}$, with N_a the total number of airports.

3.1 Introduction

Airline crew scheduling represents an important class of optimization problems where the topological structure is important. In ref. [1], a novel Potts artificial neural network approach is developed for attacking semi-realistic airline crew scheduling problems of the following type: A weekly flight schedule is given in terms of a set of flights, each with a specified airport and time for departure and arrival. The object is to assign a crew to each flight, while minimizing a cost-function defined by the total required crew time (including the waiting-time at airports). The solution is subject to a set of global constraints: The crews are required to travel along closed tours, starting and ending at a certain airport, the home-base. These tours are subject to limitations as to duration and leg-count.

As shown in ref. [1], a great deal of simplification is gained by reformulating the problem as that of mapping arrivals on departures at each airport, implying an implicit representation of the crews. In fact, without the global restrictions, the problem is reduced to a set of independent local subproblems, one at each airport. Each local problem amounts to minimizing the local waiting-time, and is simply solvable in polynomial time.

In this paper, we focus on this kind of unrestricted local problems, in particular their statistical properties. These are quite interesting, and in no way trivial, in spite of the triviality of the problems. In particular, we consider the ensemble of *random* local problems of a fixed size N, as defined by the number of arrivals/departures. In addition, we analyze the properties of random solutions to such problems.

Such a statistical analysis of this type of problem does not exist in the literature, and we feel it is interesting for the following reasons: The results illuminate the structure of the corresponding restricted problems, and as a by-product, useful tools are provided for probing their difficulty, and for simplifying their solution. Some of the methods used are novel, and may be used also in other contexts, where a similar structure occurs. In addition, a lower bound to the waiting-time is provided by the solutions to the unrestricted problem. This bound is often saturated [1].

The methodology we use contains the following steps. First, the analysis of a local problem is simplified by considering its topology (defined by the relative ordering in time between arrivals and departures) separately from its geometry (defined by the lengths of the time intervals between consecutive events). The ensemble of problems is thus factorized into the direct product of the *ensemble of topologies* and the *ensemble of geometries*.

After introducing a notation for the topology, we consider problems with a fixed topology, and evaluate averages over the geometry of entities related to the waiting-time. These are simple, since the effect of the geometry on the waiting-time spectrum is a mere shift.

The apparent difficulty of a problem is probed by analyzing the distribution of waiting-times of random solutions. A nice feature is that the waiting-time spectrum for each problem is quantized in steps of the basic period of the schedule.

Subsequently, all variables of interest are averaged also over the topology. This is a more difficult task, and requires the use of a subtle recursive method.

An alternative measure of the difficulty of a problem is the ground-state degeneracy, i.e. the number of solutions with minimal waiting-time, as compared to the total number of solutions, given by $N!$. This is independent of geometry. Due to the character of the dependence on topology, non-standard methods are required to compute the average over topology.

This paper is organized as follows: In Section 2, the problem ensemble is defined. In Section 3, a formalism is introduced for characterizing the topology. The degeneracy structure as a function of topology is analyzed, and various energy moments for fixed topology are computed, by averaging over geometry and/or pairing. In Section 4, a statistical analysis of the detailed degeneracy structure is considered. In particular, the average ground-state degeneracy is computed. Section 5 contains our conclusions.

3.2 Unrestricted Crew Scheduling

3.2.1 The Local Problem

A local problem of size N is defined by specifying the times for N arrivals (*arr*'s) and N departures (*dep*'s), denoted respectively by t_i^a and t_i^d, $i = 1 \ldots N$. The object is simply to find a one-to-one mapping (a *pairing*) between the *arr*'s and *dep*'s, such that the *energy* (or objective function) E, given by the total waiting-time, is minimal. In general this can be done in more than one way, implying a degeneracy of the ground-state.

The pairing of an *arr* A with a *dep* D implies that the crew assigned to A should next be assigned to D. The periodicity of the schedule implies that any *arr* may be mapped on any *dep*: if the *dep* is earlier, it is taken as the same *dep* in the next period.

In what follows, we will use the period as the unit of time (and thus energy). Then the times for the *arr*'s and *dep*'s can be restricted to the unit interval. If an *arr* i is mapped on a *dep* j, their contribution to the total waiting-time is given by

$$t_{ij}^w = \left(t_j^d - t_i^a\right) \bmod 1 \ \in [0,1]. \tag{3.1}$$

Thus, whatever the mapping, the energy E is restricted to the interval $[0, N]$, and between different pairings it can only change by an integer amount. Pairings yielding the lowest possible energy E_0 are said to belong to the ground-state, while a pairing with $E = E_0 + k,\ k > 0$ is said to belong to k:th excited state. In figure 3.1, an example of a flight

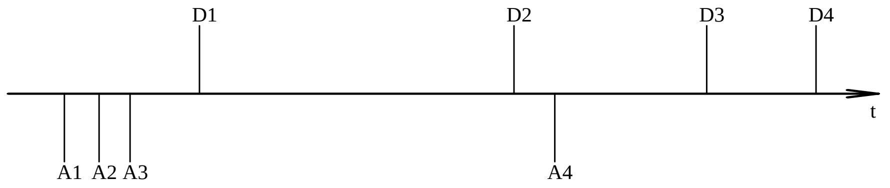

Figure 3.1: An example of a local flight schedule.

schedule is depicted.

3.2.2 The Random Local Problem

A random local problem is defined by independently choosing the times for N *arr*'s and N *dep*'s randomly on the unit period. It is useful to divide the characterization of a given problem in two parts: its **topology** and its **geometry**. The topology is defined by the relative ordering in the combined (cyclic) sequence of *arr*'s and *dep*'s. The probability is the same for every distinct topology. The geometry is defined by the sizes of the $2N$ inter-spaces $x_i, i = 1, \ldots, 2N$, into which the period is divided.

For a given problem, the solution space is defined by the set of **pairings**, i.e. the $N!$ possible mappings between *arr*'s and *dep*'s.

We will be interested mainly in the statistical properties of the following entities: the ground-state energy (minimal waiting-time) E_0, the energy E of a random pairing, and the (integer) difference $D = E - E_0$ defining the excitation energy. We will also be interested in the degeneracies of the ground-state and the excited states. To that end, we will consider three kinds of averages: over respectively the *topology*, the *geometry*, and the *pairing*.

The degeneracies of the ground-state and the excited states depend only on the topology, i.e. the combined ordering of *arr*'s and *dep*'s.

For a fixed topology, the excitation energy D is independent of the geometry, and depends entirely on the pairing. Conversely, the ground-state energy E_0 obviously is independent of the pairing, and depends only on the geometry. Thus, for a fixed topology, D and E_O are completely *uncorrelated*, in the combined ensemble of random geometries and random pairings.

3.3 Analysis for Fixed Topology

3.3.1 Characterization of the Topology

A simple way to achieve a ground-state pairing (i.e. solve the problem) for a given topology is as follows:

1. An *arr* immediately followed by a *dep* is paired with that *dep*, and both are removed from the sequence. Note that the *dep* could be in the next period.

2. The process is continued until all *arr*'s and *dep*'s are used.

As an example, consider the sequence $[A_1A_2A_3D_1D_2A_4D_3D_4]$, corresponding to the topology of the schedule in figure 3.1.

- Pairing A_3 with D_1 leaves $[A_1A_2D_2A_4D_3D_4]$.

- Pairing A_2 with D_2 leaves $[A_1A_4D_3D_4]$.

- Pairing A_4 with D_3 leaves $[A_1D_4]$.

- Pairing A_1 with D_4 finishes the process.

A graphical representation of the topology is now defined as follows.

- If necessary, rotate the sequence such that no pairing crosses the interval border.

- Represent the *arr*'s and *dep*'s by equally spaced points on the interval.

- For each pairing in turn, draw a line from the *arr* to the *dep* on the lowest level (one). All previously drawn lines that overlap with the new line are lifted one level.

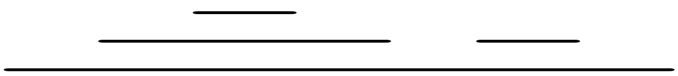

Figure 3.2: Graphical representation of the topology $[AAADDADD]$

Thus a set of lines at different levels are obtained, each line starting at an *arr* and ending on a *dep*. For the example above, the result is shown in fig. 3.2. Each line represents a crew waiting for a *dep*.

It turns out that for the properties of the total energy spectrum, full knowledge of the topology is not necessary; it suffices to know how many lines there are at the different levels. Thus let P_k be the number of lines at level k $(k = 1, 2, 3, \ldots)$. For the example above, the P sequence is $[1, 2, 1]$, i.e. $P_1 = 1, P_2 = 2, P_3 = 1$, while $P_k = 0$ for $k > 3$.

3.3.2 Degeneracy Structure

The ground-state degeneracy is now given by the product of the line-levels, i.e.

$$g_0 = \prod_{k \geq 1} k^{P_k}. \tag{3.2}$$

This is because each *dep* must terminate a line, and the number of available crews is equal to the number of lines alive at that point, which is given by the line's level. For the example we get $3 \times 2 \times 2 \times 1 = 12$, corresponding to half of the 24 possible pairings.

Naively, the degeneracy of the mth excited state can be obtained by adding m dummy lines, covering the entire interval; they represent extra crews. Then each *dep* has m additional crews to choose between, and we get for the naive degeneracy[4]

$$a_m = \prod_{k \geq 1} (k + m)^{P_k}, \tag{3.3}$$

[4]For the example above, this precisely corresponds to the spectrum $\mathbf{O}|n\!>= a_n|n\!>$ of the quantum-mechanical operator $\mathbf{O} = \mathbf{aaaa}^\dagger\mathbf{a}^\dagger\mathbf{aa}^\dagger\mathbf{a}^\dagger$, where $\mathbf{a}^\dagger, \mathbf{a}$ are harmonic oscillator creation and annihilation operators, satisfying $[\mathbf{a}, \mathbf{a}^\dagger] = 1$, and $|n\!> \propto \left(\mathbf{a}^\dagger\right)^n |0\!>$ is the n:th excited state.

where we have assumed indistinguishable dummy lines; otherwise we would have an extra factor $m!$. This defines an infinite sequence.

However, some of the possible pairings will contain permanently grounded crews (closed lines), or lines extending over more than one period. The contribution to the naive multiplicity from solutions with n grounded crews and/or excessive periods depends on the proper degeneracy n steps down. Denoting the **proper degeneracies** by g_m, the relation is

$$a_m = \sum_{n=0}^{m} g_{m-n} \begin{pmatrix} N+n \\ n \end{pmatrix}, \tag{3.4}$$

where the last factor is a binomial coefficient. This represents a kind of renormalization, and can be inverted to yield the proper degeneracies

$$g_m = \sum_{n=0}^{m} a_{m-n} (-)^n \begin{pmatrix} N+1 \\ n \end{pmatrix}. \tag{3.5}$$

This must define a finite sequence, since $g_m \geq 0$ and the total number of pairings, $\sum_m g_m = N!$, is finite.

From eq. (3.3), it is obvious that the naive degeneracy a_m is an Nth degree polynomial in m; hence it can be written as

$$a_m = \sum_{k=0}^{N} c_k \begin{pmatrix} m+k \\ k \end{pmatrix}, \tag{3.6}$$

with some coefficients c_k. Define the *generating functions*

$$A(x) = \sum_m a_m x^m \ , \ \ G(x) = \sum_m g_m x^m. \tag{3.7}$$

Due to eq. (3.5), these are related by $G(x) = (1-x)^{N+1} A(x)$. Then, in terms of c_k, we have

$$A(x) = \sum_{k=0}^{N} c_k (1-x)^{-k-1}, \tag{3.8}$$

$$G(x) = \sum_{k=0}^{N} c_k (1-x)^{N-k}. \tag{3.9}$$

From this, we see that the g sequence is indeed finite: $g(m) = 0$ for all $m > N$. The individual degeneracies g_m can be obtained from G and its derivatives at $x = 0$,

$$
\begin{aligned}
g_0 &= G(0) = \sum_k c_k, && (3.10)\\
g_1 &= G'(0) = -\sum_{k=0}^{N}(N-k)c_k,
\end{aligned}
$$

etc.

3.3.3 The Excitation Energy for a Random Pairing

Conversely, the moments over g_m are readily obtained from G and its derivatives at $x = 1$,

$$
\begin{aligned}
\sum_m g_m &= G(1) = c_N, && (3.11)\\
\sum_m m g_m &= G'(1) = -c_{N-1},\\
\sum_m m(m-1)g_m &= G''(1) = 2c_{N-2},
\end{aligned}
$$

etc. In order to relate c_k to P_k, we can express the coefficients of the polynomial a_m in two different ways. From eq. (3.3) we get for the leading coefficients

$$
a_m = m^N + m^{N-1}\sum_k kP_k + \frac{m^{N-2}}{2}\left\{\left(\sum_k kP_k\right)^2 - \sum_k k^2 P_k\right\} + \ldots, \tag{3.12}
$$

while eq. (3.6) gives, upon expanding the binomial coefficients,

$$
\begin{aligned}
a_m &= \frac{m^N}{N!}c_N + \frac{m^{N-1}}{N!}\left\{\frac{1}{2}N(N+1)c_N + Nc_{N-1}\right\} && (3.13)\\
&+ \frac{m^{N-2}}{N!}\{\frac{1}{24}N(N-1)(N+1)(3N+2)c_N\\
&+ \frac{1}{2}N^2(N-1)c_{N-1} + N(N-1)c_{N-2}\} + \ldots
\end{aligned}
$$

From this we obtain the relations

$$
\begin{aligned}
c_N &= N!, \qquad (3.14)\\
c_{N-1} &= N!\left\{\frac{1}{N}\sum_k kP_k - \frac{N+1}{2}\right\},\\
c_{N-2} &= N!\{\frac{1}{2N(N-1)}\left(\sum_k kP_k\right)^2 - \frac{1}{2N(N-1)}\sum_k k^2P_k\\
&- \frac{1}{2}\sum_k kP_k + \frac{(N+1)(3N-2)}{24}\},
\end{aligned}
$$

etc. This gives (as it should) $\sum_m g_m = N!$, the number of possible pairings. In a given topology, the fraction of pairings having an energy $D = m$ steps above E_0 is given by $g_m/N!$. Thus, the first few moments of the excitation energy D for a random pairing are:

$$
\begin{aligned}
< D >_p &= \frac{1}{N!}\sum_m mg_m = \frac{N+1}{2} - \frac{1}{N}\sum_k kP_k,\\
< D^2 >_p &= \frac{1}{N!}\sum_m m^2 g_m \qquad (3.15)\\
&= \frac{1}{N(N-1)}\left(\sum_k kP_k\right)^2 - \frac{1}{N(N-1)}\sum_k k^2P_k\\
&- \frac{N+1}{N}\sum_k kP_k + \frac{(N+1)(3N+4)}{12},
\end{aligned}
$$

where $<>_p$ denotes the average over pairings for a fixed topology.

3.3.4 The Ground-State Energy in a Random Geometry

For a given topology, the ground-state energy depends on the geometry, and is given by

$$E_0 = \sum_{i=1}^{2N} k_i x_i, \qquad (3.16)$$

where x_i is the length of the ith sub-interval, and k_i the number of lines in that interval.

To perform averages over a random geometry, we need to analyze the distribution of the intervals x_i. Independently of the topology, they obey the distribution

$$dP = (2N-1)!\,\delta\left(\sum_i x_i - 1\right)\prod_{i=1}^{2N}\left(\Theta(x_i)dx_i\right). \tag{3.17}$$

For a single x_i, this implies the distribution

$$f(x_i) = (2N-1)\,(1-x_i)^{2N-2}\ ,\ 0 \le x_i \le 1. \tag{3.18}$$

Using the identity $\sum_i x_i = 1$, and the permutation symmetry between different x_i, we have

$$\begin{aligned} < x_i > &= \frac{1}{2N}, & (3.19)\\ < x_i x_j > &= \frac{1+\delta_{ij}}{2N(2N+1)}. & (3.20)\end{aligned}$$

In what follows, we will also need the number of intervals with k lines, to be denoted by Q_k; in terms of P_k (defining $P_0 = 0$) it is simply

$$Q_k = P_k + P_{k+1},\ k \ge 0, \tag{3.21}$$

since adding a line on level k implies adding two intervals, with respectively k and $k-1$ lines. Obviously, we have $\sum_k Q_k = 2N$.

Now we are ready to compute the first few moments of E_0, yielding

$$\begin{aligned} < E_0 >_g &= \sum_{i=1}^{2N} k_i < x_i >_g = \frac{1}{2N}\sum_i k_i = \frac{1}{2N}\sum_k kQ_k \\ &= \frac{1}{N}\sum_k kP_k - \frac{1}{2}, \\ < E_0^2 >_g &= \sum_{ij} k_i k_j < x_i x_j > = \frac{1}{2N(2N+1)}\sum_{ij} k_i k_j (1+\delta_{ij}) \\ &= \frac{1}{2N(2N+1)}\left(\sum_{kl} klQ_kQ_l + \sum_k k^2 Q_k\right) \qquad (3.22) \\ &= \frac{1}{2N(2N+1)}\{4\left(\sum_k kP_k\right)^2 + 2\sum_k k^2 P_k \\ &- 2(2N+1)\sum_k kP_k + N(N+1)\}, \end{aligned}$$

where $<>_g$ denotes average over the geometry for fixed topology.

3.3.5 The Total Energy

The total energy E is given by the ground-state and excitation energies, $E = E_0 + D$. Averaging over both geometry and pairing, we obtain the surprisingly simple result

$$< E >_{gp} = < E_0 >_g + < D >_p = \frac{N}{2}, \tag{3.23}$$

independently of topology.

Since we already have the second moments of E_0 and D, we only need $< E_0 D >$ to compute the second moment of E. Using the fact that E_0 and D are statistically independent for a fixed topology, we get

$$\begin{aligned} < E_0 D >_{gp} &= < E_0 >_g < D >_p && (3.24) \\ &= -\frac{1}{N^2}\left(\sum_k kP_k\right)^2 + \frac{N+2}{2N}\sum_k kP_k - \frac{N+1}{4}. \end{aligned}$$

This implies

$$\begin{aligned} < E^2 >_{gp} &= \frac{2(N^2+N+1)}{N^2(N-1)(2N+1)}\left(\sum_k kP_k\right)^2 + && (3.25) \\ &- \frac{3(N+1)}{N(N-1)(2N+1)}\sum_k k^2 P_k + \frac{(N+1)(6N^2-N+4)}{12(2N+1)}. \end{aligned}$$

3.4 Topology Statistics

3.4.1 The Topology Ensemble

We now want to perform averages also over the topology, for the moments of the ground-state energy E_0, the excitation energy D, and the total energy E. In table 3.1 a complete list of topologies for N up to four is given, along with various characteristics. Note the similarity in symmetry-properties between the P, Q and g sequences.

To compute $< P_k >$, we must analyze the number of ways $m[P]$ to obtain a given P sequence.

- The lowest level lines define P_1 non-overlapping subgroups; these are cyclically indistinguishable, so the naive multiplicity must be divided by P_1.
- For the P_k indistinguishable lines on a higher level k, there are P_{k-1} possible lines on the previous level to put them above. This can be done in $\begin{pmatrix} P_k + P_{k-1} - 1 \\ P_k \end{pmatrix}$ ways.
- In addition, there are $2N$ cyclic rotations of the AD sequence that yield the same topology.
- The P sequence must sum up to N, and P_1 must not vanish.

This gives for the multiplicity $m[P]$

$$m[P] = \frac{2N}{P_1}\delta_{N-\sum P_k}\prod_{k=1}^{\infty}\begin{pmatrix} P_k + P_{k+1} - 1 \\ P_{k+1} \end{pmatrix}. \tag{3.26}$$

The total number of possible topologies for a fixed N is simply the number of possible AD sequences, i.e. the number of distinct orderings of a sequence of N A's and N D's. This is given by $\begin{pmatrix} 2N \\ N \end{pmatrix}$. This should match the total multiplicity $M_N \equiv \sum_{[P]} m([P])$, which can be recursively obtained from $m[P]$ by expanding the Kronecker delta in terms of a complex integral along a *small* contour C around the origin:

$$\delta_{N-\sum P_k} = \oint_C \frac{dz}{2\pi i z^{N+1}}\prod z^{P_k}. \tag{3.27}$$

By iteratively using the identity

$$z^a\sum_{b=0}^{\infty}\begin{pmatrix} a+b-1 \\ a-1 \end{pmatrix} w^b = z^a\sum_{b=0}^{\infty}\begin{pmatrix} -a \\ b \end{pmatrix}(-w)^b = \left(\frac{z}{1-w}\right)^a. \tag{3.28}$$

when summing $m[P]\prod z^{P_k}$ over the P_k in order of decreasing k, we obtain convergence of the power factor to ω^{P_k}, with ω given by

$$\omega = \frac{z}{1-\omega} \;\Rightarrow\; \omega = \frac{1}{2}\left(1-\sqrt{1-4z}\right). \tag{3.29}$$

Thus, the total multiplicity is given by

$$\begin{aligned} M_N &= 2N \oint_C \frac{dz\ z^{-N}}{2\pi i z} \sum_{P_1} \frac{\omega(z)^{P_1}}{P_1} \qquad (3.30) \\ &= 2N \oint_C \frac{dz\ z^{-N}}{2\pi i z} \left(-log(1-\omega(z))\right) = \binom{2N}{N}, \end{aligned}$$

obtained by extracting the z^N coefficient of $-\log(1-\omega(z))$ as $\frac{1}{2N}\binom{2N}{N}$, by doing the integral in terms of w using $z = w(1-w)$.

3.4.2 Distribution of P_1

The number of non-overlapping groups of lines is given by the number of lines at level 1, i.e. P_1. In fact, $w(z)$ can be seen as a generating function,

$$\begin{aligned} w(z) &= \sum_{n=1}^{\infty} m_n^{(1)} z^n = -\frac{1}{2}\sum_{n=1}^{\infty}(4z)^n \binom{n-\frac{3}{2}}{n} \qquad (3.31) \\ &= \frac{1}{2}\sum_{n=1}^{\infty}(2z)^n \frac{(2n-3)!!}{n!}, \end{aligned}$$

for the number of ways ($m_n^{(1)}$) to arrange n lines in such a group. The distribution of P_1 is easy to compute by slightly modifying eq. (3.30): skipping the summation over P_1 and dividing by M_N yields

$$\mathrm{Prob}(P_1) = \frac{2N}{\binom{2N}{N}} \oint_C \frac{dz\ z^{-N}}{2\pi i z}\frac{\omega(z)^{P_1}}{P_1} = \frac{N!(2N-1-P_1)!}{(2N-1)!(N-P_1)!} \tag{3.32}$$

for $P_1 > 0$. For large $P_1 \ll N$, this is close to 2^{-P_1}.

The grouping of lines corresponds to a grouping of the flights, with a matching number of *arr*'s and *dep*'s in each group. In a ground-state pairing, flights in different groups are never paired, and within a group, the *arr* has to precede its paired *dep*. This fact can be used to simplify also restricted problems.

3.4.3 Moments of P_k

Similarly, $< P_k >$ can be obtained from realizing that inserting a factor P_k in the sum over P_k gives a factor $P_{k-1}\omega/(1-\omega)$. This yields in the end an extra factor $P_1(\omega/(1-\omega))^{k-1}$ in the P_1 sum:

$$M_N < P_k > \; = \; 2N \oint_C \frac{dz}{2\pi i z^{N+1}} \left(\frac{\omega}{1-\omega}\right)^{k-1} \sum_{P_1>1} \omega^{P_1} \tag{3.33}$$

$$= \; 2N \oint_C \frac{dz}{2\pi i z^{N+1}} \left(\frac{\omega}{1-\omega}\right)^{k} \tag{3.34}$$

$$= \; 2N \oint_C \frac{d\omega(1-2\omega)}{2\pi i \omega^{N-k+1}(1-\omega)^{N+k+1}}, \tag{3.35}$$

where the last expression is a reformulation in terms of a loop integral around $\omega = 0$. In a similar way $< P_k P_l >$ etc. can be computed. We obtain

$$< P_k > \; = \; \frac{2N}{M_N}\left\{\left(\begin{array}{c} 2N-1 \\ N-k \end{array}\right) - \left(\begin{array}{c} 2N-1 \\ N-k-1 \end{array}\right)\right\} \tag{3.36}$$

$$= \; \frac{2k}{M_N}\left(\begin{array}{c} 2N \\ N-k \end{array}\right),$$

$$< P_k P_m > \; = \; \frac{2N}{M_N}\left\{\left(\begin{array}{c} 2N \\ N-m \end{array}\right) - \left(\begin{array}{c} 2N \\ N-k-m \end{array}\right)\right\}, \; k \leq m.$$

From these expressions, we can derive the following particular averages, needed to compute the various energy moments over the topology:

$$\left\langle \sum_k k P_k \right\rangle \; = \; \frac{N}{2} 4^N \left(\begin{array}{c} 2N \\ N \end{array}\right)^{-1} \approx \frac{N}{2}\sqrt{\pi N}\left(1 + \frac{1}{8N} + \ldots\right),$$

$$\begin{aligned}
\left\langle \sum_k k^2 P_k \right\rangle &= N^2, \\
\left\langle \left(\sum_k k P_k \right)^2 \right\rangle &= \frac{N^2(5N+1)}{6},
\end{aligned} \tag{3.37}$$

where the approximate form in the top equation is valid for large N.

3.4.4 Full Energy Averages

We are now ready to compute the final averages also over the topology. Inserting the results of eqs. (3.37) into eqs. (3.22), we obtain for the moments of the ground-state energy E_0 of a random problem:

$$\begin{aligned}
< E_0 > &= \frac{1}{2} 4^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-1} - \frac{1}{2} \approx \frac{1}{2}\sqrt{\pi N} - \frac{1}{2}, \\
< E_0^2 > &= \frac{5N}{6} - \frac{1}{2} 4^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-1} + \frac{1}{2} \approx \frac{5N}{6}, \\
< E_0^2 >_c &\approx \frac{(10-3\pi)N}{12},
\end{aligned} \tag{3.38}$$

where $\langle ab \rangle_c = \langle ab \rangle - \langle a \rangle \langle b \rangle$ is the connected moment; the approximate forms are valid for large N.

Similarly, from eqs. (3.15) we get for the moments of the excitation energy D, for a random pairing in a random topology,

$$< D > = \frac{N+1}{2} - \frac{1}{2} 4^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-1} \approx \frac{N}{2} - \frac{1}{2}\sqrt{\pi N} + \frac{1}{2} \tag{3.39}$$

$$\begin{aligned}
< D^2 > &= \frac{N^2}{4} + \frac{17N}{12} + \frac{1}{3} - \frac{N+1}{2} 4^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-1} && (3.40) \\
&\approx \frac{N^2}{4} - \frac{N}{2}\sqrt{\pi N} + \frac{17N}{12}, \\
< D^2 >_c &\approx \frac{(11-3\pi)N}{12}. && (3.41)
\end{aligned}$$

Averaging the combined moment, eq. (3.24), over topology yields

$$
\begin{aligned}
< E_0 D > &= \frac{N+2}{4} 4^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-1} - \frac{13N+5}{12} && (3.42) \\
&\approx \frac{N}{4}\sqrt{\pi N} - \frac{13N}{12}, \\
< E_0 D >_c &= -\frac{5N+1}{6} + \frac{1}{4} 16^N \begin{pmatrix} 2N \\ N \end{pmatrix}^{-2} && (3.43) \\
&\approx -\frac{(10-3\pi)N}{12}.
\end{aligned}
$$

Combining this with the moments of E_0 and D, we get for the moments of the total energy, $E = E0 + D$, the following simple results

$$
\begin{aligned}
< E > &= \frac{N}{2}, && (3.44) \\
< E^2 > &= \frac{N^2}{4} + \frac{N}{12}, && (3.45) \\
< E^2 >_c &= \frac{N}{12}, && (3.46)
\end{aligned}
$$

which can be understood by noting that a random pairing in a random topology corresponds to a set of N lines with random endpoints. Then each line has a uniform length distribution between 0 and 1, and E is their total length.

Computing the corresponding standard deviations, we have for a typical random problem (and a random pairing, for D and E) at large N,

$$
\begin{aligned}
E_0 &\sim \frac{1}{2}\sqrt{\pi N} \ \pm\ \frac{1}{2}\sqrt{\left(\frac{10}{3} - \pi\right) N}, && (3.47) \\
D &\sim \frac{N}{2} - \frac{1}{2}\sqrt{\pi N} \ \pm\ \frac{1}{2}\sqrt{\left(\frac{11}{3} - \pi\right) N}, && (3.48) \\
E &\sim \frac{N}{2} \ \pm\ \frac{1}{2}\sqrt{\frac{N}{3}}, && (3.49)
\end{aligned}
$$

and we see that E scales as $\sqrt{N}$, while E and D scale as N, while the standard deviation scales as $\sqrt{N}$ in each case.

Of interest are also the correlations at large N, given to order N by

$$\begin{aligned} < E_0 D >_c &\sim -\frac{N}{12}(10 - 3\pi), \\ < E_0 E >_c &\sim 0, \\ < DE >_c &\sim \frac{N}{12}, \end{aligned} \tag{3.50}$$

indicating that E and E_0 become uncorrelated for a random pairing of a large random problem.

3.4.5 Statistics for Individual Degeneracies

An interesting but more difficult thing to compute is the average fraction $\gamma_n =< g_n > /N!$ of pairings having a given excitation energy $D = n$, in particular the ground-state fraction γ_0, which gives the average probability of hitting a ground-state by chance. Since g_n is simply related to a_n, we will start by considering

$$\alpha_n = \frac{\begin{pmatrix} 2N \\ N \end{pmatrix}}{2N} < a_n >, \tag{3.51}$$

in terms of which γ_n can be expressed as

$$\gamma_n = \frac{N!}{(2N-1)!} \sum_m (-)^m \begin{pmatrix} N+1 \\ m \end{pmatrix} \alpha_{n-m}. \tag{3.52}$$

We then have

$$\begin{aligned} \alpha_n &= \frac{1}{2N} \sum_{[P]} m[P] \prod_k (n+k)^{P_k} \\ &= \sum_{[P]} \frac{1}{P_1} \delta_{N - \sum P_k} \prod_{k \geq 1} (n+k)^{P_k} \begin{pmatrix} P_k + P_{k+1} - 1 \\ P_{k+1} \end{pmatrix}. \end{aligned} \tag{3.53}$$

A generating function for the N-dependence of α_n is then

$$A_n(z) \equiv \sum_N z^N \alpha_n(N) = \sum_{[P]} \frac{1}{P_1} \prod_{k \geq 1} [z(n+k)]^{P_k} \begin{pmatrix} P_k + P_{k+1} - 1 \\ P_{k+1} \end{pmatrix}. \tag{3.54}$$

Again, starting the P_k summation at a large k_0 and proceeding in order of decreasing k, gives convergence of the full sum in the limit $k_0 \to \infty$. Denoting by $\omega_{k+n-1}^{P_k}$ the result of summing above a certain k, we have by eq. (3.28),

$$\omega_{k-1} = \frac{kz}{1 - w_k}, \tag{3.55}$$

and the final result is

$$A_n(z) = \sum_{P_1} \frac{1}{P_1} \omega_n^{P_1} = -\log\left(1 - \omega_n(z)\right). \tag{3.56}$$

The recursion relation (3.55) can now be linearized by assuming $\omega_k = p_k/q_k$, which can be solved e.g. by

$$q_{k-1} = q_k - p_k, \tag{3.57}$$
$$p_{k-1} = kzq_k, \tag{3.58}$$

which, upon eliminating p_k gives

$$kzq_k - q_{k-1} + q_{k-2} = 0. \tag{3.59}$$

By partial integration, it is simple to prove that the following sequence of integrals solves the recursion relation (3.59) for q_k,

$$q_k = \mathrm{Im} \int_0^\infty \frac{(-iy)^k}{k!} \exp\left(-z\frac{y^2}{2} + iy\right) dy\ ,\ k \geq 0,\ z > 0, \tag{3.60}$$

which can be extended to negative k by recursion. The q_k can be expanded in z as

$$q_k = \begin{cases} \sum_{m=0}^{\infty} \begin{pmatrix} 2m+k \\ 2m \end{pmatrix} (2m-1)!!z^m\ ,\ k \geq 0, \\ \sum_{m=0}^{[-(k+1)/2]} \begin{pmatrix} -k-1 \\ 2m \end{pmatrix} (2m-1)!!z^m\ ,\ k < 0, \end{cases} \tag{3.61}$$

where [] denotes integer part. Note that for non-negative k the series is an asymptotic one, while for negative k it is finite. In particular, we have $q_{-1} = 1$. An independent solution to (3.59) is given by

$$\hat{q}_k(z) = \frac{z^{-k}}{k!} q_{-k-1}(-z), \tag{3.62}$$

but this solution is irrelevant, having the wrong large-k behaviour.

In terms of q_k we now have

$$\begin{aligned} 1 - \omega_k &= q_{k-1}/q_k, & (3.63) \\ A_n &= \log(q_n) - \log(q_{n-1}), & (3.64) \end{aligned}$$

which should then be expanded in powers of z to yield $\alpha_n(N)$ as the coefficient of z^N. In particular, we have $A_0 = \log(q_0)$, where $q_0 = 1 + z + 3z^2 + 15z^3 + \ldots$.

In table 3.2, results are displayed for the average degeneracy of the two lowest states, based on an expansion of A_0 and A_1 in powers of z. It is easy to check for small N, using table 3.1, that $g_m/\sum g_k$ for $m = 0, 1$, averaged over topologies with the proper probabilities, indeed agrees with γ_m of table 3.2, obtained from the expansion of A_k.

The result for γ_0 strongly indicates an asymptotic behaviour of $\gamma_0 \sim 2N2^{-N}$. This corresponds to an exponential decrease with N of the average probability for a random pairing to hit a ground-state. However, the average number of ground-states grows faster than exponentially: $< g_0 > \sim 2N\ N!\ 2^{-N}$.

This abundance of ground-states indicates that a corresponding restricted problem might well have a solution with a locally minimal waiting-time, if the restrictions are not too severe; this is used in ref. [1] to simplify the solution of a set of restricted crew scheduling problems.

By reducing the state-space of a restricted problem to the set of ground-states of the corresponding unrestricted problem, the average information gained at each airport is given by $\log \gamma_0$, which for a large airport roughly yields $N \log 2$. Summing this over several airports yields a total information gain scaling as $N_f \log 2$, with N_f the total number of flights. Partly, this is due to a grouping of flights, which contributes an average information gain of $N_a \log 2$, with N_a the number of airports.

3.5 Conclusions

We have performed a statistical analysis of an ensemble of random unrestricted local crew scheduling problems, formulated in terms of mapping arrivals onto departures at a single airport so as to minimize waiting-time.

For the ground-state energy E_0 of a large random problem, we find that both the average and the fluctuations scale like $\sqrt{N}$. For a random pairing of such a problem, on the other hand, the excitation D and the total energy $E = E_0 + D$ both grow linearly with N, with fluctuations scaling like $\sqrt{N}$.

The individual degeneracies of the lowest energy states for random problems are such that the average probability for hitting an optimal pairing by chance decreases like $2N2^{-N}$ for large N. Since the total number of pairings grows like $N!$, the average number of ground-states grows very fast with system size.

The results and the methods of analysis described in this paper are useful when designing efficient algorithms for the crew scheduling problems with global restrictions, by providing means for estimating the difficulty of a given problem, and for understanding and simplifying its structure.

The optimal crew waiting-time for a restricted problem is bounded from below by the ground-state energy of the corresponding unrestricted problem, which is useful for gauging algorithmic performance for problems of realistic size. Due to a faster than exponential growth of the number of ground-states with problem size, this bound is often saturated. This can be used to simplify a restricted problem: By insisting on the local ground-state energy, airports can be split into several parts. For a large random problem, this results on the average in a reduction of state-space size by a factor of two for each airport.

Some of the calculations in this paper are based on novel methods of a general nature, that may have applications also in other contexts with a similar topological structure.

N	Pattern	Prob.	$\{P_k\}$	$\{Q_k\}$	$\{a_i\}$	$\{g_i\}$	$\langle E_0 \rangle$	$\langle D \rangle$
1	AD	1	[1]	[1,1]	1,2,3,...	(1)	1/2	0
2	AADD	2/3	[1,1]	[1,2,1]	2,6,12,...	(2)	2/2	0
2	ADAD	1/3	[2]	[2,2]	1,4,9,...	(1,1)	1/2	1/2
3	AAADDD	3/10	[1,1,1]	[1,2,2,1]	6,24,60,...	(6)	9/6	0
3	AADADD	3/10	[1,2]	[1,3,2]	4,18,48,...	(4,2)	7/6	1/3
3	AADDAD	3/10	[2,1]	[2,3,1]	2,12,36,...	(2,4)	5/6	2/3
3	ADADAD	1/10	[3]	[3,3]	1,8,27,...	(1,4,1)	3/6	3/3
4	AAAADDDD	4/35	[1,1,1,1]	[1,2,2,2,1]	24,120,360,...	(24)	8/4	0
4	AAADADDD	4/35	[1,1,2]	[1,2,3,2]	18,96,300,...	(18,6)	7/4	1/4
4	AAADDADD	4/35 }	[1,2,1]	[1,3,3,1]	12,72,240,...	(12,12)	6/4	2/4
4	AADAADDD	4/35 }						
4	AAADDDAD	4/35	[2,1,1]	[2,3,2,1]	6,48,180,...	(6,18)	5/4	3/4
4	AADADADD	4/35	[1,3]	[1,4,3]	8,54,192,...	(8,14,2)	5/4	3/4
4	AADDAADD	2/35 }	[2,2]	[2,4,2]	4,36,144,...	(4,16,4)	4/4	4/4
4	AADADDAD	4/35 }						
4	AADDADAD	4/35	[3,1]	[3,4.1]	2,24,108,...	(2,14,8)	3/4	5/4
4	ADADADAD	1/35	[4]	[4,4]	1,16,81,256,...	(1,11,11,1)	2/4	6/4

Table 3.1: Inequivalent topologies for $N \leq 4$. For each topology, the third column gives the probability of occurrence, while the next two give its P_k and Q_k sequence, respectively. In the following two columns the a_i and g_i sequences are given, whereas the last two columns give the average ground-state energy and excitation, respectively. Note that different topologies might have identical characteristics in terms of P_k, etc.

N	$K_N = \frac{1}{2}\binom{2N}{N}$	$K_N < g_0 >$	$K_N < g_1 >$	$< g_0 >$	$< g_1 >$	γ_0	γ_1
1	1	1	0	1.00000	0.00000	1.000000	0.000000
2	3	5	1	1.66667	0.33333	0.833333	0.166667
3	10	37	22	3.70000	2.20000	0.616667	0.366667
4	35	353	411	10.0857	11.7429	0.420238	0.489286
5	126	4081	7676	32.3889	60.9206	0.269907	0.507672
6	462	55205	149741	119.491	324.115	0.165960	0.450159
7	1716	854197	3.09875×10^{6}	497.784	1805.80	0.098767	0.358294
8	6435	1.4876×10^{7}	6.84187×10^{7}	2311.74	10632.3	0.057335	0.263697
9	24310	2.88019×10^{8}	1.61447×10^{9}	11847.7	66411.7	0.032649	0.183013
10	92378	6.13891×10^{9}	4.07031×10^{10}	66454.3	440614.	0.018313	0.121421
11	352716	1.42882×10^{11}	1.09496×10^{12}	405092.	3.10436×10^{6}	0.010148	0.077771
12	1.35208×10^{6}	3.60668×10^{12}	3.13708×10^{13}	2.66751×10^{6}	2.32019×10^{7}	0.005569	0.048438
13	5.20030×10^{6}	9.81584×10^{13}	9.55147×10^{14}	1.88755×10^{7}	1.83672×10^{8}	0.003031	0.029496
14	2.00583×10^{7}	2.86562×10^{15}	3.08337×10^{16}	1.42865×10^{8}	1.5372×10^{9}	0.001639	0.017633

Table 3.2: Results for the average degeneracy of the lowest energy-states for various system sizes N. The second column gives an integer normalization factor K_N. Dividing the integers in the next two columns by K_N yields the average number of ground-states $< g_0 >$ and first excited states $< g_1 >$, respectively. Dividing these by $N!$ yields γ_0 and γ_1.

Bibliography

[1] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling using Potts Mean Field Techniques", *LU TP 97-10* (submitted to *European Journal of Operational Research*).

Chapter 4

A Potts Neuron Approach to Communication Routing

Paper IV

LU TP 97-2
Second Revision
March 30, 1998

A Potts Neuron Approach to Communication Routing

Jari Häkkinen[1], Martin Lagerholm[2], Carsten Peterson[3] and Bo Söderberg[4]

Complex Systems Group,
Department of Theoretical Physics University of Lund,
Sölvegatan 14A, SE-223 62 Lund, Sweden

Neural Computation **10**, (1998)

[1]jari@thep.lu.se
[2]martin@thep.lu.se
[3]carsten@thep.lu.se
[4]bs@thep.lu.se

Abstract:

A feedback neural network approach to communication routing problems is developed, with emphasis on *Multiple Shortest Path* problems, with several requests for transmissions between distinct start- and endnodes. The basic ingredients are a set of Potts neurons for each request, with interactions designed to minimize path lengths and to prevent overloading of network arcs. The topological nature of the problem is conveniently handled using a propagator matrix approach. Although the constraints are global, the algorithmic steps are based entirely on local information, facilitating distributed implementations. In the polynomially solvable single-request case, the approach reduces to a fuzzy version of the *Bellman-Ford* algorithm. The method is evaluated for synthetic problems of varying sizes and load levels, by comparing to exact solutions from a branch-and-bound method, or to approximate solutions from a simple heuristic. With very few exceptions, the Potts approach gives legal solutions of very high quality. The computational demand scales merely as the product of the numbers of requests, nodes, and arcs.

4.1 Introduction

Communication routing resource allocation problems are becoming increasingly relevant given the upsurge in demand of the Internet and other telecommunication services. One such problem amounts to assigning arcs in a connected network to requests from start- to endnodes, given capacity constraints on the arcs, such that a total additive cost (path-length) is minimized. For a review of notation and existing routing techniques, see e.g. ref. [1]. A relatively simple routing problem, with only one request at a time, is the Shortest Path Problem (**SPP**), which can be solved exactly in polynomial time using e.g. the Bellman-Ford (**BF**) algorithm [2] (see also [1]). The Multiple Shortest Path Problem (**MSPP**), where arcs are to be allocated simultaneously to several requests, is more difficult. We strongly suspect MSSP to be NP-hard, although to our knowledge this has not been proven in the literature.

In this paper we address the MSPP using feedback Potts neural networks, which have proven to be powerful in other resource allocation problems, with [3] or without [4] a non-trivial topology. For each request we assign a Potts network, with units encoding which arcs to be used by that request. Appropriate energy terms are constructed in terms of the Potts neurons to minimize total path lengths and to ensure that capacity constraints are not violated. Mean field (**MF**) equations are iterated using annealing to minimize the total energy. In contrast to earlier usage of Potts encoding and MF annealing [5, 4, 3] where global objective functions are minimized, here each node minimizes its own local energy.

For the case of a *single* request, the Potts MF approach reduces in the zero temperature limit to the BF algorithm; hence our approach contains this standard algorithm as a special case.

For each request the Potts MF network [5] defines an (inverted) "fuzzy" *spanning tree*[5], rooted at the endnode. In order to project out the part defining the (fuzzy) path from the startnode, and to keep track of the paths in general, we utilize a propagator matrix formalism following

[5] A subgraph without loops connecting all nodes.

ref. [3]. The computation of the propagator requires matrix inversion; fortunately this can be done using an iterative procedure with a low computational cost.

As in a previously considered airline crew scheduling problem [3], proper preprocessing is employed to identify independent subproblems, in order to reduce the problem complexity.

Despite the existence of global constraints, the implementation of the approach is truly local – when updating the MF equations for a particular node, only information residing at neighbouring nodes is needed.

The approach is gauged by an exact branch-and-bound (**BB**) algorithm (for smaller problems), and by two BF-inspired heuristics (for larger problems), on a set of synthetic but challenging test problems, showing an excellent performance of the Potts MF approach, with a CPU consumption per request scaling merely like NN_L, where N is the number of nodes and N_L the number of links in the network. The method is also very robust with respect to parameters.

Another novel method [6] has been proposed for the MSSP, which in contrast to our approach is aimed at dynamical problems. It is also rooted in the BF algorithm, but is in the static limit unrelated to our Potts algorithm. However, in this limit it reduces to the independent BF approach, which is used for comparisons in this work.

4.2 The Multiple Shortest Path Problem

A Multiple Shortest Path Problem (MSPP) is defined by specifying the following:

- A connected network of N nodes and N_L (bidirectional) links, corresponding to $2N_L$ arcs.
- For each arc ij, a cost (arc-length) d_{ij} and a capacity C_{ij}.
- A set of N_R transmission requests r, each defined by a startnode a_r and an endnode b_r.

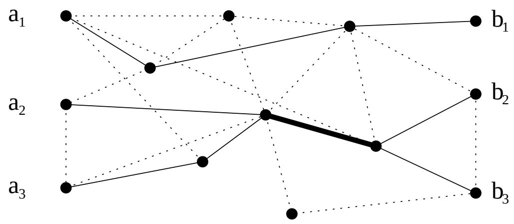

Figure 4.1: Example of a solution to a 3-request problem. Dotted lines represent unused links, and full lines links used by the requests.

The task is then to assign to each request a connected loop-free path of arcs from the startnode to the endnode. This is to be done such that the total cost for those paths is minimized, without the load on any arc exceeding its capacity, with the load defined as the number of requests sharing it. A 3-request problem example is shown in fig. 4.1.

Not all problems of this kind are solvable; a reliable algorithm should be able to recognize and signal a failure, to enable proper measures to be taken.

4.3 The Bellman-Ford Algorithm in the Mean Field Language

Prior to dealing with the MSPP, we revisit the simpler SPP and demonstrate how the BF algorithm can be recast in a Potts MF language. This formulation will then be the starting point for designing a Potts MF approach to MSPP.

In the SPP there is only a single request, from a to b, and the capacity constraints are irrelevant. The task simply is to find the shortest path from a to b. In the BF algorithm [1] this is achieved by minimizing the path-lengths D_i from every node i to b, by iterating

$$D_i \rightarrow \min_j (d_{ij} + D_j), \quad i \neq b. \tag{4.1}$$

and keeping track of the involved arcs ij. D_b is fixed to zero by definition.

The resulting solution defines a spanning tree rooted at b. In particular, D_a is determined, and the minimal path from a to b is easily extracted from the spanning tree.

If no arc exists from node i to j, d_{ij} could formally be defined to be infinite; in practice it is more convenient to restrict j in eq. (4.1) to the actual *neighbours* of i, reachable via an arc from i.

Eq. (4.1) can be rewritten as

$$D_i = \sum_j v_{ij} E_{ij} \equiv \sum_j v_{ij}(d_{ij} + D_j) \tag{4.2}$$

in terms of a *Potts spin* $\mathbf{v}_i$ for every node $i \neq b$, with components v_{ij} taking the value 1 for the j with the smallest *local energy* E_{ij}, and 0 for the others (winner-takes-all). Note the distinct philosophy here: Each node i minimizes its own local energy $D_i = \min_j E_{ij}$, rather than all nodes striving to minimize some global objective function.

A *mean field* (MF) version of eq. (4.2) is obtained by using for $\mathbf{v}_i$ its thermal average in the MF approximation, defined by

$$v_{ij} = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}} \tag{4.3}$$

where j and k are neighbours of i, and T is an artificial temperature. Note that each *Potts MF neuron* $\mathbf{v}_i$ obeys the normalization condition

$$\sum_j v_{ij} = 1, \ i \neq b \ . \tag{4.4}$$

Thus, v_{ij} can be interpreted as a *probability* for node i to choose j as a continuation node.

At a non-zero temperature, iteration of eqs. (4.2,4.3) can be viewed as a fuzzy implementation of the BF algorithm, while in the $T \to 0$ limit the neurons are forced *on-shell*,[6] i.e. $v_{ij} \to 1$ (for the minimizing j) or 0 (for the rest), and proper BF is recovered.

[6]This notation originates from physics, where "on-shell" and "off-shell" denote whether particles are real or virtual.

Given this obvious neural recast of the BF algorithm in terms of Potts neurons, it is somewhat surprising that non-exact neural approaches based on Ising spins have been advocated in the literature [7].

4.4 The Potts Mean Field Approach to MSPP

The Potts MF formulation of the Bellman-Ford algorithm for SPP (eqs. (4.2,4.3)) is a suitable starting point for approaching MSPP.

We will stick with the philosophy inherited from BF of focusing on independent *local energies*, in contrast to what has become standard when using feedback neural networks for resource allocation problems. This represents a novel strategy.

Thus, we introduce a separate Potts system, $\{v_{ij}^r\}$, for each request r, with basic local energies E_{ij} as before representing distances to the endnode b_r. In addition, we will need energy terms E_{ij}^{load} for the load-constraints, to be discussed later; this introduces an interaction between the Potts systems.

This formulation introduces the possibility of undesired loop formation, since forming a loop might induce less energy penalty than violating a load constraint. As will be discussed below, such loops can be suppressed by suitable penalty terms, and by adding a possibility for each proper node to connect, via an artificial *escape arc*, to an artificial *escape node*, for each request connecting directly to the endnode. This enables a "give-up" state for an unresolvable situation, signaled by some path containing the escape node. The cost for "giving up" must be larger than that for any legal path. Therefore the cost of each escape arc is set to the sum of the costs of the proper arcs, while the corresponding capacity is chosen large enough to be able to host all the requests.

In order to terminate the path for a request r, its endnode must be a *sink* for the corresponding Potts system. Consequently, there will be no Potts neuron $\mathbf{v}_{b_r}^r$ associated with it.

In order to construct appropriate penalty terms, a propagator matrix

Figure 4.2: Graphical illustration of the expansion, eq. (4.5), of the propagator $\mathbf{P}^r$ with respect to path-length. A dot ($\bullet$) represents a node i, and a line (—) a single node-to-node step v^r_{ij}.

will be used. This technique has proven to be a powerful tool in neural optimization for problems with a non-trivial topology [3]. In particular, it will be crucial for extracting properties of the fuzzy paths defined by the MF approach at finite T.

4.4.1 Path Extraction and the Propagator

The normalization condition (eq. (4.4)) ensures that for each request r, precisely one continuation is chosen for each node except b_r, although for $T \neq 0$ it is fuzzily distributed over the available neighbours. On shell, the path from start- to endnode is trivial to extract – one follows the $v^r_{ij} = 1$ path starting from the startnode. However, for $T \neq 0$ a more refined path extraction mechanism is needed. This is provided by a *propagator matrix* [3] $\mathbf{P}^r$ for each request r, defined by:

$$\begin{aligned} P^r_{ij} &= \left[(\mathbf{1} - \mathbf{v}^r)^{-1}\right]_{ij} = \delta_{ij} + v^r_{ij} + \sum_k v^r_{ik} v^r_{kj} \\ &+ \sum_{kl} v^r_{ik} v^r_{kl} v^r_{lj} + \ldots \end{aligned} \tag{4.5}$$

For a graphical representation, see fig. 4.2. On shell, it is easy to see that P^r_{ij} can be interpreted as the *number of paths* from i to j defined by the Potts neurons associated with r; similarly, the elements of the matrix square $(P^r)^2_{ij}$ are related to the number of arcs used in those paths[7].

[7] More precisely, the number of arcs is given by $P^2 - P$.

In particular, we have in the absence of loops ($\Rightarrow P^r_{ii} \equiv 1$),

$$P^r_{a_r i} = \begin{cases} 1, & \text{if node } i \text{ appears in the path } a_r \to b_r, \\ 0, & \text{otherwise,} \end{cases}$$

identifying the arcs used in the paths.

Off shell, these interpretations still are valid in a probabilistic sense. Thus, a probabilistic measure of how much node i participates in the path $a_r \to b_r$ is given (naively) by $P^r_{a_r i}$ (since $P^r_{ib_r} \equiv 1$)[8]. Dividing by P^r_{ii} to correct for a possible normalization error due to a small loop-contamination, yields the *path-factor*

$$F^r_i \equiv \frac{P^r_{a_r i}}{P^r_{ii}} \leq 1 \ , \tag{4.6}$$

that can be used to filter out the nodes of the fuzzy paths defined by the MF neurons.

4.4.2 Load and Loop Control

Armed with the propagator formalism, we proceed to set up the penalty terms, to be added to the energies E_{ij} corresponding to eq. (4.2).

In order to efficiently avoid complications from self-interactions in the local energies, independently of T, we follow ref. [8], and define the penalty terms based on analyzing the result of setting one component v^r_{ij} of the neuron $\mathbf{v}^r_i$ at a time to one, with the other components set to zero, as compared to a reference state with all components set to zero.

The total load L_{ij} on an arc ij is the sum of contributions from the different requests,

$$L_{ij} = \sum_r L^r_{ij}. \tag{4.7}$$

The contribution from request r can be expressed as

$$L^r_{ij} = F^r_i v^r_{ij} \tag{4.8}$$

[8] The path must eventually end in the sink b_r. *Proof*: Multiply eq. (4.4) for $\mathbf{v}^r$ by P^r_{ki}, sum over i, and use the matrix identity $\mathbf{P}^r = \mathbf{1} + \mathbf{P}^r\mathbf{v}^r$.

where we have used the path-factor of eq. (4.6).

The load constraints, $L_{ij} \leq C_{ij}$, define a set of inequality constraints. In the realm of feedback neural networks, such constraints have been successfully handled by means of step-functions [8]. For a given request r, the overloading of the arc ij due to the other requests is given by

$$O(X) \equiv X\Theta(X) , \tag{4.9}$$

where $\Theta()$ is the Heaviside step-function, and X is given by

$$X \equiv L_{ij} - L_{ij}^r - C_{ij} . \tag{4.10}$$

If the arc where also to be used by r, the overloading would increase to $O(X+1)$, and a suitable overloading penalty can be defined as the difference

$$E_{ij}^{\text{load}} = O\left(X+1\right) - O\left(X\right) . \tag{4.11}$$

The amount of loops introduced by connecting $i \rightarrow j$ can be expressed as

$$Y \equiv \frac{P_{ji}^r}{P_{ii}^r} \leq 1 , \tag{4.12}$$

and we choose as a loop suppression term

$$E_{ij}^{\text{loop}} = \frac{Y}{1-Y}. \tag{4.13}$$

The generalization of the local energy in eq. (4.2) to the multiple request case now reads, for a particular request r,

$$E_{ij} = d_{ij} + D_j^r + \alpha E_{ij}^{\text{load}} + \gamma E_{ij}^{\text{loop}} \tag{4.14}$$

with the added terms based on eqs. (4.11,4.13). The resulting algorithm allows for a wide range of choices of the coefficients α and γ without severely changing the performance.

4.4.3 Updating Equations

All neurons are repeatedly updated, with a slow annealing in T. For each request r and each node i, the corresponding neuron $\mathbf{v}_i^r$ is updated according to

$$v_{ij}^r = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}}. \tag{4.15}$$

with E_{ij} given by eq. (4.14). The corresponding cost D_i^r from node i to the endnode is then updated, in the BF spirit, as

$$D_i^r \rightarrow \sum_j v_{ij}^r E_{ij}. \tag{4.16}$$

In principle, the corresponding update of the propagator could be done using an exact incremental matrix inversion scheme like Sherman–Morrison [9]. We prefer, though, to let local changes propagate through the network, in analogy to the update of D_i^r. Thus, only the i'th row of $\mathbf{P}^r$ is updated:

$$P_{im}^r \rightarrow \delta_{im} + \sum_j v_{ij}^r P_{jm}^r \text{ , for all } m \tag{4.17}$$

This gives a convergence towards the exact inverse, which turns out to be good enough. The advantage of this method is twofold: it is faster, and all information needed is local to the relevant node i and its neighbours j (assuming each node to keep track of its own row of $\mathbf{P}^r$).

Details of the algorithmic steps and the initialization can be found in Appendix A.

4.5 Test Problems and Explorations

In order to test the Potts MF method we have generated a set of challenging synthetic problems. The most important parameters governing the difficulty of a problem, apart from network size and connectivity, are the number of requests and the average arc capacity. In cases where all the arcs are able to host all the requests, $C_{ij} \geq N_R$, the problem is

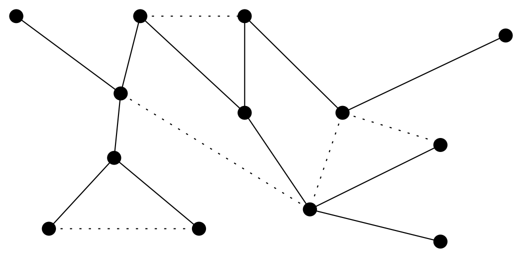

Figure 4.3: An example network with 13 nodes and 17 links. The solid lines define the initial spanning tree, and the dotted lines represent the additional links.

separable into an independent SPP for each request, that can be solved using the BF algorithm. We have therefore chosen to work with rather tight arc capacities.

For each problem, we generate a random connected network, where every node has at least one path to all other nodes. To that end, all nodes are first connected in a random spanning tree. Additional links (creating loops) are then randomly placed. Every arc is given a random integer capacity between 1 and 6, and a random cost in the interval (0,1). The desired number of random requests is generated, in terms of start- and end-nodes. An example of a generated test problem is shown in fig 4.3.

This procedure does not automatically yield a solvable problem, where all requests can be fulfilled simultaneously without violating any constraint. In principle, solvability could be built into the problem-generator, but here we adopt another strategy. *Small problems* (see table 4.1) we attempt to solve exactly with a branch-and-bound algorithm (BB); those not solved within a certain amount of CPU time are disregarded. In principle, this method could introduce a bias towards simple problems. However, for the smaller problem sizes considered, only a tiny fraction of the problem candidates are "timed-out", except for those defining the last row in table 4.1, where about 1/3 are disregarded. For *larger problems* (see table 4.2), BB can not be used, and we do not know in general whether they are solvable or not.

Prior to attacking a problem, a decomposition into independent sub-problems is attempted, to reduce complexity. The decomposition utilizes the fact that a network typically consists of sub-networks, connected to each other via single nodes. The complete problem can then be solved by independently solving the implied sub-problem for each sub-network.

The CPU demand of the non-reduced MF approach scales like $N_R N N_L$, while the required computer time for a decomposed problem is dominated by the largest sub-problem. In table 4.2 we indicate the effect on the CPU demand of the decomposition for the large test problems.

The performance of the MF Potts approach is probed by measuring the relative excess path length as compared to a reference algorithm (X),

$$\Delta_X = \frac{D_{\mathrm{MF}} - D_{\mathrm{X}}}{D_{\mathrm{X}}}, \tag{4.18}$$

where D_{MF} and D_{X} are the total path costs resulting from MF and the reference algorithm, respectively.

4.5.1 Comparison with an Exact Method

For the smaller problems the Branch-and-Bound algorithm is used as a reference algorithm. The results are displayed in table 4.1, together with characteristics of the generated test problems. As a measure of the complexity of a problem we use the entropy, S, defined as the logarithm of the total number of possible configurations, disregarding load constraints. Table 4.1 indicates an excellent performance of the MF Potts approach, with respect to giving rise to legal solutions with good quality, with a very modest computational demand.

4.5.2 Comparing with Other Approximative Methods

The computer demand for the BB algorithm grows rapidly. For problems larger than those of table 4.1 we have instead as reference algorithms chosen two BF-inspired heuristics:

Nodes	Links	Requests	$\langle S\rangle$	legal$_{\mathrm{MF}}$	$\langle\Delta_{\mathrm{BB}}\rangle$	$\langle$CPU-time$\rangle$
5	10	5	14	100.0%	0.003	0.1
5	10	10	28	99.9%	0.002	0.2
10	15	10	28	100.0%	0.004	0.4
10	20	10	48	100.0%	0.003	0.5
15	20	10	27	99.8%	0.03	0.5
15	20	15	40	99.9%	0.06	0.7

Table 4.1: Results for small problems, 1000 problem of each size are probed. Only legal entries are used when calculating the averages $\langle\Delta_{\mathrm{BB}}\rangle$ and $\langle$CPU-time$\rangle$. $\langle$CPU time$\rangle$ refers to the MF Potts approach using a DEC Alpha 2000, and is given in seconds. Typical times for the BB method are around 600 seconds.

- **Independent Bellman-Ford (IBF)**. Each request is taken to define an independent SPP, which is solved using BF. Thus, the load constraints are neglected, and the resulting solution is not necessarily legal; if legal, it gives the global energy minimum.
- **Sequential Bellman-Ford (SBF)**. The requests are served in a sequential fashion using the BF algorithm. When the maximum capacity of an arc is reached, its use is prohibited for the remaining requests. It does not always find a solution, even to a solvable problem, and when it does, it is not necessarily the global minimum.

The early ARPANET [1] (Internet) protocol was based on an algorithm similar to IBF (but with a dynamically updated arc cost based on traffic congestion).

The results are displayed in table 4.2. In the few cases where the global minimum is known (from IBF), it is found by the MF-algorithm. In the other cases, we conclude from table 4.2 that MF is on the average performing better than SBF. When the MF cannot find a solution, this is signaled by requests routed via the escape node.[9] In all such cases,

[9] This could be used in dynamical cases as a decision tool for which requests should be put on hold.

N	N_L	N_R	Red.	$\text{legal}_{\text{IBF}}$	$\langle\Delta_{\text{IBF}}\rangle$	$\text{legal}_{\text{SBF}}$	$\langle\Delta_{\text{SBF}}\rangle$	legal_{MF}
50	200	20	1.00	4%	0.000	100%	-0.019	100%
80	200	20	0.97	8%	0.000	98%	-0.021	98%
100	200	20	0.91	7%	0.000	96%	-0.015	96%
100	200	50	0.90	0%	——	82%	-0.043	82%
100	200	100	0.91	0%	——	22%	-0.092	30%
100	300	100	0.99	0%	——	83%	-0.066	84%
100	400	100	1.00	0%	——	99%	-0.063	100%

Table 4.2: Results for large problems, 100 problems of each kind are probed. The averages, $\langle\Delta_X\rangle$, are for legal entries. The reduction (Red.) is defined as $N_R N N_L$ after the decomposition divided by the value before the decomposition. Typical CPU times for MF are $O(10)$ minutes on a DEC Alpha 2000.

also IBF and SBF have failed.

4.6 Summary and Outlook

We have developed a Potts MF neural network algorithm for finding approximate solutions to the Multiple Shortest Path Problem. The starting point is a Potts MF recast of the exact Bellman-Ford algorithm for the simpler single Shortest Path Problem. This approach is then extended to the Multiple Shortest Path Problem by utilizing several Potts networks, one for each request.

Complications of topological nature are successfully handled by means of a convenient propagator approach, which is crucial for the following issues:

- The MF approach yields at $T \neq 0$ fuzzy spanning trees, from which the propagator is used to extract the loads corresponding to the fuzzy paths, needed for the interaction between different requests.
- Loops are suppressed by energy terms, based on the propagator.

To open up escape paths for unresolvable situations, an auxiliary arc to an escape node is introduced for each proper node.

The method is local in that only information available from neighbouring nodes is required for the updates. This attractive feature, inherited from the Bellman-Ford algorithm, facilitates a distributed implementation.

The computational demand of the method is modest. The CPU time scales as $N_R N N_L$. With fixed connectivity this corresponds to $N_R N^2$, whereas for the worst case of full connectivity it yields $N_R N^3$.

The performance of the algorithm is tested on a set of synthetic challenging problems. This is done by (i) comparing to exact results from a branch-and-bound method for smaller problems and (ii) comparing to results from independent and sequential Bellman-Ford approaches for larger problems. The comparisons show that the Potts MF approach with very few exceptions yields very good approximate solutions at modest computer time consumption.

Presently, the method is being generalized to other routing problems [10].

Appendix A

The Potts MF Algorithm

A.0.1 Initialization

The initial temperature T_0 is first set to $T_0 = 50$. If the saturation Σ,

$$\Sigma \equiv \frac{1}{N_R(N-1)} \sum_{i \neq b_r} \mathbf{v}_i^2, \tag{A.1}$$

has changed more than 10% after all neurons have been updated once, then the system is reinitialized with $T_0 \rightarrow 2T_0$.

For all nodes (except the end- and escape nodes), the corresponding Potts neurons are initialized in accordance with the high temperature limit, i.e.

$$v_{i,j}^r = 1/n_i \tag{A.2}$$

for all n_i neighbours j (including the escape node) of i. P_{ij}^r and D_i^r are initialized consistently with eq. (A.2).

A.0.2 Iteration

Until $T \leq T_f$ or $\Sigma \geq \Sigma_f$ do:

- For every request r do:
  1. For every node i except b_r and the escape node:
     (a) Update $\mathbf{v}_i^r$ (eqs. (4.3,4.14)).
     (b) Update D_i^r (eq. (4.16)).
     (c) Update $\mathbf{P}_i^r$ (eq. (4.17)).
  2. Update L_{ij}.
- Decrease the temperature: $T = kT$.

We have used the parameter values: $k = 0.9$, T_f=0.0001, Σ_f=0.99999. For the energy coefficients in eq. (4.14), we have consistently used $\alpha = 1$ and $\gamma = 5$.

Bibliography

[1] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, New Jersey (1987).

[2] R. Bellman, "On a Routing Problem", *Quarterly of Applied Mathematics* **16** 87 (1958).

[3] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling with Potts Neurons", *Neural Computation* **9**, 1627 (1997).

[4] L. Gislén, C. Peterson and B. Söderberg, "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, 805 (1992).

[5] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[6] J.A. Boyan and M.L. Littman, "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach", in *Advances in Neural Information Processing Systems* **6**, J.D. Cowan, G. Teauso and J. Alspector, eds., Morgan-Kauffman, San Fransisco (1994).

[7] S.C.A. Thomopolous, L. Zhang and C.D. Wann, "Neural Network Implementation of the Shortest Path Algorithm for Traffic Routing in Communication Networks", *Proceedings of the IEEE International Joint Conference on Neural Networks*, Singapore, 2693 (1991).

[8] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints – the Knapsack Problem", *Neural Computation* **5**, 331 (1993).

[9] See e.g. W.P. Press, B.P Flannery, S.A. Teukolsky and W.T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, Cambridge (1986).

[10] J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg, in preparation.

Chapter 5

Local Routing Algorithms Based on Potts Neural Networks

Paper V

LU TP 98-5
February 28, 1998

Local Routing Algorithms Based on Potts Neural Networks

Jari Häkkinen[1], Martin Lagerholm[2], Carsten Peterson[3] and Bo Söderberg[4]

Complex Systems Group,
Department of Theoretical Physics University of Lund,
Sölvegatan 14A, SE-223 62 Lund, Sweden

Submitted to *IEEE/ACM Transactions on Networking*

[1]jari@thep.lu.se
[2]martin@thep.lu.se
[3]carsten@thep.lu.se
[4]bs@thep.lu.se

Abstract:

A feedback neural approach to static communication routing in asymmetric networks is presented, where a mean field formulation of the Bellman-Ford method for the single unicast problem is used as a common platform for developing algorithms for *multiple unicast*, *multicast* and *multiple multicast* problems. The appealing locality and update philosophy of the Bellman-Ford algorithm is inherited. For all problem types the objective is to minimize a total connection cost, defined as the sum of the individual costs of the involved arcs, subject to capacity constraints. The methods are evaluated for synthetic problem instances by comparing to exact solutions for cases where these are accessible, and else with approximate results from simple heuristics. The computational demand is modest.

5.1 Introduction

Static routing problems amount to assigning arcs (edges) to a set of communication requests in a network, such that a total arc cost is minimized subject to capacity constraints on the arcs. For a review of such problems and existing routing techniques, see e.g. [1]. In this paper we develop a family of novel distributed algorithms for static routing problems with asymmetric links, based on a *Potts neuron* encoding and a mean field (**MF**) relaxation dynamics. Such approaches have proven powerful in many resource allocation problems [2], including cases with a non-trivial topology [3].

For the relatively simple *single unicast* (shortest path) problem, several fast exact methods exist, scaling polynomially with the network size, e.g. Bellman-Ford (**BF**) [4], Dijkstra and Floyd-Warshall [1]. For other problem types one in general has to rely on various heuristics [5]. Here, we will exploit a recast of BF in a neural form, with a Potts MF neuron at each node [6], as a starting point for approaching more complex problems, with the appealing local update philosophy of BF preserved. For problems with multiple simultaneous requests, a separate Potts network is assigned to each request, interacting through penalty terms encoding load constraints.

The *multiple unicast* problem, where arcs are to be allocated simultaneously to several unicast requests, is probably NP-hard, although we are not aware of any rigorous proof for this. Preliminary results using the Potts MF approach for this problem were reported in [6].

The generic *single multicast* problem, where one message is to be sent to several receivers, is known to be NP-complete [7]. The arc capacities are irrelevant, and the objective is simply to minimize the sum of the individual costs of the arcs used.

In a *multiple multicast* problem the capacity constraints come into play, and a load interaction has to be introduced in the dynamics.

The performance of a Potts MF method for each of these problems is evaluated by a comparison to different approximate schemes. For multiple unicast simple heuristics based on BF are used, and for multicast a Directed Spanning Tree Heuristic (**DSTH**), inspired by the Minimal Spanning Tree Heuristic [5], is employed. A heuristic based on a sequential application of DSTH is used for multiple multicast. For small enough problems, an exact Branch-and-Bound (**BB**) method is used to provide an exact solution.

Despite the global nature of the problems, the implementation of the Potts approach is truly local – when updating the MF neurons for a particular node, only information residing at neighbouring nodes is needed. Together with a good performance, this represents a key asset of the method.

This paper is organized as follows. In Section 5.2 the problem types are defined and discussed. A section each on the three problem types then follow, where the corresponding Potts approach is described and evaluated. A brief summary can be found in Section 5.6. Algorithmic details on the Potts MF algorithms, the BB algorithms and a minimal directed spanning tree heuristic are given in Appendices A, B and C, respectively.

5.2 Problem Discussion

5.2.1 Networks

In the problem types considered, a *network* is assumed to be given, defined by a connected graph of N nodes and L (bidirectional) links, corresponding to $2L$ arcs, each with a specified cost (arc-length) and capacity.

We will limit our interest to networks with at most one link for every node pair. Hence an arc from node i to node j can be unambiguously labeled by the ordered pair of node labels (ij); this notation improves readability of formulae and will be used throughout the paper, with the restriction to linked node pairs being understood.

For an arc (ij), its cost d_{ij} could represent e.g. an actual cost, or the delay of a signal traveling through the arc, while its (integer) capacity $C_{ij} > 0$ represents the maximum number of simultaneous signals it can hold.

5.2.2 Problem Types

On a given network, a *unicast* request is defined by specifying a *sender* node a, that is to transmit a message to a *receiver* node b. Similarly, a *multicast* request is defined by specifying a sender node a, with a single message aimed at several receiver nodes $b_i, i = 1 \ldots B$.

A *multiple unicast* problem is then defined by specifying a set of simultaneous single unicast requests, $a_r, b_r, r = 1 \ldots R$, that are to be routed, such that the total path length is minimized, without any arc capacity being exceeded.

In a *single multicast* problem, a multicast request is to be routed through the shortest directed tree that is rooted at the sender and reaches all the receivers. The length of the tree is defined as the sum of the arc lengths of the used arcs.

Finally, in a *multiple multicast* problem several multicast requests are given. Each should be routed, such that the sum of the resulting tree lengths is minimized, while respecting arc capacities.

5.2.3 Random Problems

To gauge the various algorithms for the three problem types, we have used artificial random problems, defined on a pool of artificially generated random networks described in Table 5.1.

To ensure that a network with N nodes and L links is connected, it is built by first using $N-1$ links to create a random spanning tree. Then each remaining link is used to connect a random, previously unconnected pair of nodes; the two corresponding (oppositely directed) arcs are independently assigned a random cost in the interval $[0, 1]$, and a random integer capacity in the range $\{1, \ldots, 6\}$.

When a network of the desired size is chosen from the pool, a random problem is constructed by generating the desired number of independent random requests, for the multicast case with the desired number of receiver nodes.

5.2.4 Problem Reduction

The topology of a network might admit a decomposition into subnetworks, such that a routing problem reduces to a set of independent subproblems, each in its own subnetwork.

Every node that is such that its removal will disconnect the network defines a *split-node*. At every such node, the network can be split in two or more parts, each with its own replica of that node. By splitting the network at all split-nodes, a tree of subnetworks, connected via the split-nodes, will be formed; Fig. 5.1 shows an example.

To keep track of the relation between the original problem and the resulting subproblems, an auxiliary graph, to be referred to as the *hypertree*, can be defined, containing the original nodes as well as the subnetworks as formal nodes, with links representing the belonging of a node to a subnetwork, as depicted in Fig. 5.1c. For a routing problem in a reducible network, the corresponding trivial problem in the hypertree is first solved; its unique solution determines the decomposition of the given problem into subproblems.

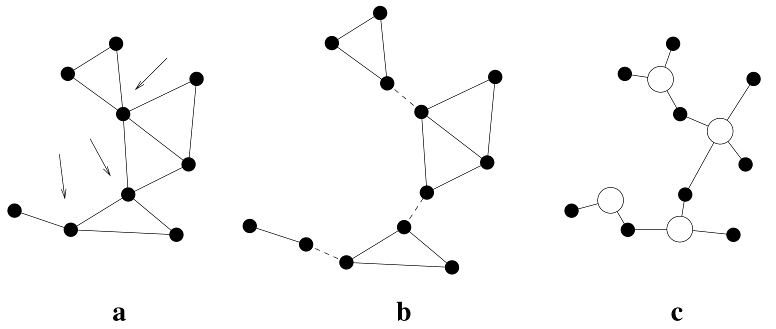

Figure 5.1: **(a).** A network with the split-nodes marked with arrows. **(b).** The decomposition into subnetworks. **(c).** The hypertree; every second node (white blobs) represents a subnetwork.

The complexity of a problem can be taken as the size of the space of possible routings; this depends on the problem and is in general difficult to compute. Instead, noting that the Bellman-Ford algorithm scales as $(N-1)L$, we will use a rough, but simple and problem-independent measure Q of the reduction in network complexity, defined as follows. With N_s and L_s denoting the number of nodes and links in a subnetwork s, we simply define

$$Q \equiv \frac{\sum_s (N_s - 1)\, L_s}{(N-1)L} \equiv \frac{\sum_s (N_s - 1)\, L_s}{\sum_s (N_s - 1) \sum_s L_s} \leq 1. \tag{5.1}$$

Table 5.1 shows the average reduction factors for the pool of random networks. The reduction is most important for the exact BB methods, where the actual time reduction factor can be considerable, even for $Q \approx 1$.

N	L	$< Q >$
5	10	1.00
10	20	0.97
15	20	0.61
20	30	0.75
50	100	0.91
50	200	1.00
100	200	0.90
100	400	1.00
200	250	0.42

Table 5.1: Sizes chosen for the pool of random networks. 1000 networks of each size are generated. Also shown is the average reduction factor.

5.3 Multiple Unicast

In this section we focus on developing a Potts mean field algorithm for the multiple unicast problem. This is done in two steps: In Subsection 5.3.1 the Bellman-Ford algorithm for the single unicast problem is recast into a Potts mean field language, while in Subsection 5.3.2 an extension to and tools to handle the multiple case is described. The solution to a 3-request problem is shown in Fig. 5.2.

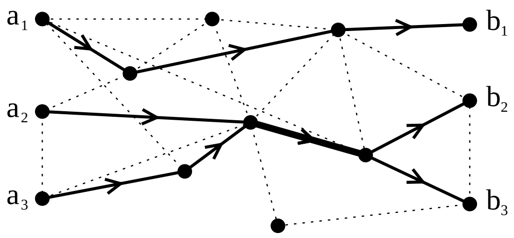

Figure 5.2: Example of a solution to a 3-request unicast problem. Dotted lines represent unused links, and full lines links that are used by the requests.

5.3.1 Mean Field Version of the Bellman-Ford Algorithm

For a *single* unicast, the arc capacities are irrelevant, and the task is simply to find the shortest path from a sender a to a receiver b. In the BF algorithm [4], this is done by relaxing, for each node i, the estimated shortest path-length D_i to b, according to

$$D_i \to \min_j (d_{ij} + D_j) \equiv \min_j E_{ij},\ i \neq b, \tag{5.2}$$

and keeping track of the chosen neighbours j. Note the distinct philosophy here: Each node i minimizes its own *local energy* E_{ij}, rather than all nodes striving to minimize some global objective function.

Eq. (5.2) can be written as

$$D_i = \sum_j v_{ij} E_{ij} \tag{5.3}$$

in terms of a *winner-take-all neuron* $\mathbf{v}_i$ for every node $i \neq b$, with components v_{ij} taking the value 1 for the optimal neighbour j, and 0 for the others.

A *mean field* (MF) version of BF is obtained by replacing the discrete winner-take-all neurons by MF Potts neurons, with components given by

$$v_{ij} = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}}\ , \tag{5.4}$$

where T is an artificial temperature. The component v_{ij} is to be interpreted as a *probability* for node i to choose neighbour j as a continuation node, consistently with $\sum_j v_{ij} = 1$.

At a non-zero temperature, iteration of (5.3, 5.4) can be viewed as a soft (fuzzy) version of the BF algorithm. At high T, the probability is evenly distributed between the available neighbours, while in the $T \to 0$ limit a winner-take-all dynamics results, and the proper BF algorithm is recovered.

By starting at a non-vanishing T, and slowly letting $T \to 0$ (annealing), the MF neurons will gradually converge to sharp winner-take-all states,

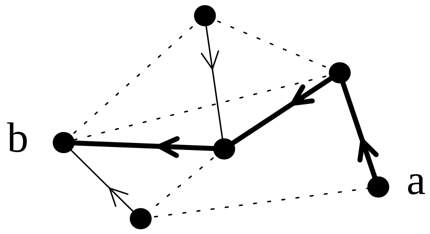

Figure 5.3: Example of a solution to a single unicast problem. The arcs of the emerging tree are shown as solid lines with arrows indicating the direction, while unused links are shown dotted. The path from **a** to **b** is found by following the arrows; it is marked with fat lines.

and a spanning tree directed towards b will emerge (a *BF-tree*), consisting of the chosen arcs. In particular, the cost D_a for the optimal path from sender to receiver is determined; the path itself is simply extracted from the spanning tree, c.f. Fig. 5.3.

The resulting MF algorithm will be referred to as **PSU** (Potts Single Unicast); it is not very interesting in itself, but will serve as a stepping stone towards MF algorithms for the more difficult problem types. Note that BF, and thus PSU, will always yield a tree solution – loops do not pay.

5.3.2 The Potts Mean-Field Approach to Multiple Unicast: PMU

The obvious generalization of PSU to a multiple unicast problem is by having a separate Potts system for each unicast request, labeled $r = 1 \ldots R$. To enforce the capacity constraints, the local energies E_{ij}^r will have to be supplemented with a penalty term for arc overloading. The competition for arcs then destroys the no-loop guarantee, since the path-choice for one request might block the way for another. To suppress potential loop-formation, an additional penalty term is added, and the local energy takes the form

$$E_{ij}^r = d_{ij} + D_j^r + \alpha E_{ij}^{r,\mathrm{load}} + \gamma E_{ij}^{r,\mathrm{loop}} \ , \tag{5.5}$$

where D_j^r is the cost (including load and loop penalties) from node j to the endnode in request r. Note that the Bellman-Ford philosophy is kept, only the energy is redefined, following [6].

In spite of the loop-suppression term, a neuron might wind up in an impossible situation, with no good continuation node available. Following [6] an escape facility is introduced, enabling an (expensive) emergency route to b_r.

To construct the load and loop terms, a *propagator* formalism is employed, following ref. [3]; for a more detailed explanation see [8]. For the Potts system managing the routing of request r, an element of the propagator $\mathbf{P}^r$ is defined as

$$P_{ij}^r = ((\mathbf{1} - \mathbf{v}^r)^{-1})_{ij} = \delta_{ij} + v_{ij}^r + \sum_k v_{ik}^r v_{kj}^r + \sum_{kl} v_{ik}^r v_{kl}^r v_{lj}^r + \ldots \quad (5.6)$$

It could be interpreted as the (fuzzy) number of paths $i \to j$ in the BF-tree for request r, and becomes integer for $T = 0$. It enables the definition of a probabilistic measure, F_i^r, of how much node i participates in the path $a_r \to b_r$ serving request r,

$$F_i^r \equiv \frac{P_{a_r i}^r}{P_{ii}^r} \frac{P_{i b_r}^r}{P_{b_r b_r}^r} = \frac{P_{a_r i}^r}{P_{ii}^r} \; (\leq 1). \quad (5.7)$$

The simpler form follows from $P_{ib_r}^r \equiv 1$. The desired penalty terms can now be defined, based on the propagator.

Thus, the load L_{ij}^r on an arc (ij) due to the request r is given by $L_{ij}^r = F_i^r v_{ij}^r \leq 1$. Summing the contributions from all requests yields the total arc-load, $L_{ij} = \sum_r L_{ij}^r$. For a particular request r, the overloading of the arc (ij) due to the other requests is given by

$$W(X) \equiv X\Theta(X) \; , \quad X \equiv L_{ij} - L_{ij}^r - C_{ij} \; , \quad (5.8)$$

where Θ is the Heaviside step function. With the arc also used by r, the overloading would increase to $W(X+1)$, and the difference will serve as an overloading penalty,

$$E_{ij}^{r,\text{load}} = W\left(X+1\right) - W\left(X\right). \quad (5.9)$$

In addition, the amount of loops introduced by connecting i to j can be expressed as the amount of path from j to i, as given by $Y \equiv P^r_{ji}/P^r_{ii}$ (≤ 1), and we choose as a loop suppression term

$$E^{r,\text{loop}}_{ij} = \frac{Y}{1-Y}. \tag{5.10}$$

With a separate Potts system for each request, the updating equations (5.3, 5.4) will be replaced by

$$D^r_i \to \sum_j v^r_{ij} E^r_{ij} \tag{5.11}$$

with

$$v^r_{ij} = \frac{e^{-E^r_{ij}/T}}{\sum_k e^{-E^r_{ik}/T}} \tag{5.12}$$

The propagators will be updated in a "soft", local manner, relaxing towards (5.6):

$$P^r_{im} \to \delta_{im} + \sum_j v^r_{ij} P^r_{jm}\ , \quad \text{for all } m. \tag{5.13}$$

The algorithm defined by iterating (5.5, 5.11, 5.12, 5.13) with annealing in T will be referred to as **PMU**; contrary to the case for PSU, it does *not* correspond to an exact algorithm in the zero temperature limit.

Note that iterating only at $T = 0$ will in general not lead to a good result, since a choice made for one request could force a sub-optimal choice for another. By using instead the mean field annealing technique, the decisions are made in an incremental way, allowing neurons to gradually form their decisions under the influence of the emerging decisions of the other neurons. Note also that the philosophy inherited from the BF algorithm is not disturbed, all information needed is local to the relevant node i and its neighbours j, with each node keeping track of its own row of $\mathbf{P}^r$.

5.3.3 Evaluation of PMU

The performance of the PMU algorithm is gauged against three other algorithms.

Independent Bellman-Ford Heuristic – IBF
Each request is independently solved using BF, disregarding the load constraints. Thus, the result might be illegal, whereas a legal result is necessarily the exact minimum. For "tight" networks (arc capacities low in relation the signal density) illegal results are in general produced.

Sequential Bellman-Ford Heuristic – SBF
Here, the unicast requests are served in a random order using BF; when the maximum capacity of an arc is reached, its use is prohibited for the subsequent requests. This algorithm can be run repeatedly, with the request order reshuffled in between, until a preset time limit is used up; then the best result is kept. It does not always find a legal result, even to a solvable problem; when it does, it is not necessarily the minimum.

Branch-and-Bound – BB
For small enough problems, an exact Branch-and-Bound algorithm, presented in Appendix B, is used to find the exact minimum.

The rate of legal results as well as their quality is probed for network sizes spanning from 5 to 100 nodes, see Table 5.2. The computational demand for BB grows very fast with the network size, therefore a CPU time limit of 5 minutes is used. For sizes where BB's chances of finding a solution within the time interval is small, it is not used at all, those entries are marked with "—" in the table. The SBF heuristic is allowed to run for a slightly longer time than PMU as specified in Table 5.2.

The quality of the results from an algorithm Y as compared with that of a reference algorithm X can be measured in terms of the relative excess path length,

$$\Delta_{\mathrm{X}}(\mathrm{Y}) = \frac{D_{\mathrm{Y}} - D_{\mathrm{X}}}{D_{\mathrm{X}}}, \tag{5.14}$$

where D_{X} is the total path costs resulting from X. Mean values for problems where both algorithms gave legal results are presented in Table 5.2.

From Table 5.2 one finds as could be expected that not all problems were solvable, and that PMU and SBF gave rise to approximately the same number of legal results. Problem instances with a legal IBF result can be regarded as easy: The minimum coincides with that of the unconstrained problem. From Table 5.2, one finds that in a region of tight problems

PMU is doing slightly better than SBF, whereas the oppposite is true in the other end.

5.4 Single Multicast

An attempt to solve a multicast problem should yield a multicast tree (**MT**), defined as a directed tree spanning the set of sources S (the sender and the receivers), rooted at the sender. A minimal MT is of course desired; arc capacities are irrelevant.

5.4.1 The Potts Mean-Field Approach to Single Multicast: PSM

The update philosophy of PSU for a single unicast problem is based on each node minimizing its estimated distance to the end node. For the multicast problem we instead adopt the strategy that each node i attempts to minimize the distance to its *forward MT*, defined as the partial MT that is spanned by those sources connecting to the sender via paths *not* passing node i. Apart from this modification, the development of PSM closely follows PSU.

PSU gives a tree directed *towards* the root, a BF-tree. We want the opposite direction, equivalent to transposing d_{ij} and C_{ij} (since the network is asymmetric). We will therefore work entirely in the transposed network, where a legal result corresponds to an MT directed *towards* b, now corresponding to the sender-node. In PSU a single path was extracted from the BF-tree; in the multicast case, a subtree corresponding to an MT should be extracted. For an example of a solution to a multicast problem in the transposed network, see Fig. 5.4.

Let D^{ia} denote the estimated distance from node i to the (fuzzy) path originating from a source node $a \in S$. It is handled by node i and calculated via the neighbours (propagated) according to

$$D^{ia} = F_{ai} \sum_j v_{ij} \left(d_{ij} + D^{ja} \right), \text{ where } F_{ai} = 1 - \frac{P_{ai}}{P_{ii}} \,. \tag{5.15}$$

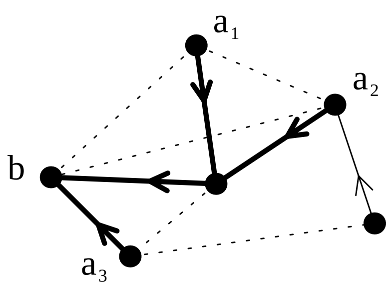

Figure 5.4: Example of an MT solving a single multicast problem in the transposed network. The nodes a_i, $i = 1, 2, 3$, are receivers and b the sender. The arcs of the emerging BF-tree are shown as solid lines with arrows, with the subset defining the MT marked with fat lines. Unused links are shown dotted.

The factor $F_{ai} \leq 1$ is a measure to what extent the path from a to b avoids node i. This factor drives D^{ia} to measure the distance from i to the path from a.

For sharp paths, the distance $\tilde{D}_{ij}$ to the forward MT of node i via a neighbour node j, could then be expressed as $d_{ij} + \min_{a \in S} D^{ja}$, restricted to a with $F_{ai} = 1$ (i not in its path). For fuzzy paths, one could use $d_{ij} + \min_{a \in S} D^{ja}/F_{ai}$, where a source a with i in its path is penalized with a large factor. Instead of entirely using D^{ja} for the best source a, however, we will use a weighted value in the MF spirit, given by

$$\tilde{D}_{ij} = d_{ij} + \frac{\sum_{a \in S} D^{ja} e^{-D^{ja}/(F_{ai}\kappa T)}}{\sum_{a \in S} e^{-D^{ja}/(F_{ai}\kappa T)}} \, , \tag{5.16}$$

where κ is a suitably tuned temperature ratio.

The neurons v_{ij}, managing the choice of neighbour, are updated according to (5.4), with the local energy redefined as

$$E_{ij} = \tilde{D}_{ij} + \gamma E_{ij}^{\text{loop}}. \tag{5.17}$$

where E_{ij}^{loop} is defined as in (5.10) but for one request, i.e. neglecting the index r[5]. A load term is not needed; arcs not capable of hosting

[5]Using the loop term will penalize loops immediately; this improves the final result.

at least one signal are ignored. Likewise, the propagator is updated in the same manner as for PMU, i.e. (5.13) neglecting the index r. The resulting algorithm will be referred to as **PSM**.

5.4.2 Evaluation of PSM

In order to evaluate the performance of PSM we have developed a *Directed Spanning Tree Heuristic, DSTH,* as a heuristic adaptation to asymmetric networks of the Minimal Spanning Tree Heuristic, MSTH [5]. Algorithmic details are given in Appendix C. For small enough problems, a BB algorithm, described in Appendix B, is used to find a minimal MT.

Table 5.3 shows the number of legal results and the average CPU consumption for the different algorithms, and a performance comparison according to (5.14 based on the MT-cost. A bar ("—") marks unexplored entries.

From Table 5.3 one finds that to all except 2 of the 9000 probed problems the PSM algorithm gave legal results, with an average quality consistently superiour to that of DSTH.

5.5 Multiple Multicast

A Potts algorithm for the multiple multicast problem is constructed in an obvious way, by extending PSM in analogy to the extension of PSU to PMU (Subsection 5.3.2). Evaluation of the resulting algorithm is done by comparing to BB and a sequential DSTH heuristic.

5.5.1 The Potts Mean-Field Approach to Multiple Multicast: PMM

As in PMU, a separate Potts system $\mathbf{v}^r$, with a corresponding propagator $\mathbf{P}^r$, is introduced for each of the R multicast requests r. The load constraints are again relevant; the load on an arc from the (fuzzy) MT of a request r is calculated as a "fuzzy OR" over the corresponding source paths,

$$L_{ij}^r = 1 - \prod_{a \in S^r} (1 - \frac{P_{ai}^r}{P_{ii}^r} v_{ij}^r) \ \leq 1 \ . \tag{5.18}$$

where S^r denotes the set of nodes defined by the sender and the receivers in request r. A penalty term is formed according to (5.8, 5.9) with the total arc load given by $L_{ij} = \sum_r L_{ij}^r$. For a loop penalty term, (5.10) is used without modification. The escape facility is defined and used in the same way as in PMU. In analogy with Subsection 5.4.1, but now for each request r, each node i attempts to minimize the distance to the corresponding forward MT with respect to the choice of neighbour j. For a fixed j, this distance is estimated as

$$\tilde{D}_{ij}^r = d_{ij} + \sum_{a \in S_r} D^{rja} \frac{e^{-D^{rja}/\left(F_{ai}^r \kappa T\right)}}{\sum_{b \in S_r} e^{-D^{rjb}/\left(F_{bi}^r \kappa T\right)}} \ , \tag{5.19}$$

where D^{ria} is the distance from i, along the fuzzy BF-tree serving request r, to the path from one of its source nodes a; it is updated as

$$D^{ria} = F_{a_r i}^r \sum_j v_{ij}^r \left(d_{ij} + D^{rja} \right) \ . \tag{5.20}$$

The local energy corresponding to request r thus takes the form

$$E_{ij}^r = \tilde{D}_{ij}^r + \alpha E_{ij}^{r,\mathrm{load}} + \gamma E_{ij}^{r,\mathrm{loop}} \ . \tag{5.21}$$

The neurons and the propagators are updated as in Section 5.3.2, i.e. using the equations (5.12) and (5.13), respectively. The resulting algorithm will be referred to as **PMM**.

5.5.2 Evaluation of PMM

The performance of the PMM approach is gauged using a Branch-and-Bound algorithm, presented in Appendix B and a *Sequential Directed Spanning Tree Heuristic, SDSTH*, an extension of the DSTH algorithm described in Appendix C: A DSTH is applied to each multicast request in turn, blocking arcs having reached their capacity. The SDSTH is used in the same way as SBF was used in the multiple unicast case; it is allowed to run for the time specified in Table 5.4.

The results for the different algorithm are shown in Table 5.4. The PMM heuristic finds approximately the same number of legal results as SDSTH ($\approx 1\%$ less), with a consistently better quality.

5.6 Summary

A family of Potts mean field feedback artificial neural network algorithms is developed and explored for artificial multiple unicast and single as well as multiple multicast routing problems.

In order to handle loads and loops for "fuzzy" paths, and other probabilistic measures, a propagator formalism is used.

The Potts approach is local for all the problem types, with all information needed by a node for an update residing at this node and its neighbours. This attractive feature, inherited from the single unicast Bellman-Ford algorithm, facilitates a distributed implementation.

The Potts algorithms are compared to competitive heuristics, and gauged against exact methods whenever feasible, with encouraging results. For large problems they perform consistently better than the other heuristics.

The CPU consumption is proportional to the product of the number of requests, receivers, nodes and links.

Another neural network method [9] has been proposed for the multiple unicast problem; in contrast to our approach it is aimed at dynamical problems. However, in the static limit it reduces to the independent BF approach, which is used for comparisons in this work.

			legal results					<CPU>		$<\Delta_{\mathrm{X}}(\textsc{pmu})>$		
N	L	R	PMU	BB	IBF	SBF	T_{BB}	PMU	SBF	BB	IBF	SBF
5	10	5	1000	1000	713	1000	0	0.1	1.0	0.002	0.000	0.00049
10	20	5	994	1000	705	994	0	0.2	1.0	0.003	0.000	0.00247
10	20	10	973	954	289	973	19	0.4	1.0	0.005	0.000	0.00341
10	20	15	936	93	91	933	839	0.7	1.0	0.004	0.001	0.00519
15	20	15	460	—	39	484	–	0.6	5.0	—	0.000	0.00654
20	30	20	508	—	5	496	–	1.6	5.0	—	0.000	-0.00364
50	200	50	996	—	0	997	–	53	100	—	—	-0.00006
100	200	100	297	—	0	313	–	268	300	—	—	-0.03903
100	400	100	988	—	0	990	–	515	600	—	—	-0.02229

Table 5.2: **Multiple unicast**. Number of legal results and CPU time used fro the different algorithms, and a quality comparison. T_{BB} is the number of instances for which BB was prematurely terminated. 1000 instances of each problem size are probed. CPU is given in seconds on an DEC Alpha 250. The quality of the PMU results is gauged by comparing to BB, IBF and SBF respectively, for instances where both algorithms produced legal result (for BB when it found a solution within the time limit); a negative $\Delta_{\mathrm{X}}(\textsc{pmu})$ indicates that PMU is superior to X.

			legal results			CPU time			$<\Delta_X(\text{PSM})>$	
N	L	B	PSM	BB	DSTH	PSM	BB	DSTH	BB	DSTH
5	10	4	1000	1000	1000	0.07	0.00	0.00	0.004	-0.062
10	20	4	1000	1000	1000	0.19	0.00	0.00	0.009	-0.056
10	20	9	999	1000	1000	0.27	0.01	0.00	0.011	-0.069
15	20	14	1000	1000	1000	0.24	0.01	0.00	0.012	-0.047
20	30	15	1000	1000	1000	0.59	0.07	0.00	0.017	-0.056
50	200	40	1000	—	1000	12.8	—	0.03	—	-0.117
100	200	99	1000	—	1000	24.8	—	0.19	—	-0.097
100	400	99	1000	—	1000	69.9	—	0.29	—	-0.127
200	250	150	999	—	1000	30.3	—	0.37	—	-0.050

Table 5.3: **Single multicast**. The number of legal results, average consumed CPU time and the relative quality of the results from the different algorithms. 1000 instances of each problem size are probed. Same notation as in Table 5.2; B denotes the number of receivers in each multicast.

				legal results				CPU usage		$< \Delta_{\mathrm{X}}(\mathrm{PMM}) >$	
N	L	R	B	BB	SDSTH	PMM	T_{BB}	PMM	SDSTH	BB	SDSTH
5	10	5	4	989	998	998	9	0.43	1	0.008	-0.064
10	20	10	4	0	755	700	722	2.03	5	—	-0.005
15	20	4	5	—	565	596	—	0.74	5	—	-0.017
20	30	5	5	—	673	671	—	2.1	5	—	-0.020
20	30	5	10	—	267	270	—	5.3	6	—	-0.019
50	100	5	25	—	321	296	—	27.5	40	—	-0.053
50	200	10	25	—	881	871	—	100	150	—	-0.118

Table 5.4: **Multiple multicast**. The number of legal results and average consumed CPU time for the different algorithms, and a quality comparison. 1000 instances of each problem size are probed. Same notation as in Tables 5.2 and 5.3.

Appendix A

PMM – Algorithmic Details

The temperature T is assigned a tentative initial value of $T_0 = 150$. Until $(\Delta v)^2 < 0.1$ after one iteration, with

$$\begin{aligned}(\Delta v)^2 &= \frac{1}{R(N-1)} \sum_{rij} (\Delta v_{ij}^r)^2 \qquad \text{(A.1)} \\ &= \frac{1}{R(N-1)} \sum_{rij} (v_{ij}^r(t+1) - v_{ij}^r(t))^2 > 0.1\end{aligned}$$

the system is reinitialized with $T_0 \rightarrow 2T_0$.

Each Potts neuron $\mathbf{v}_i^r$ is initialized by assigning equal values, with 1% random variation, to its components v_{ij}^r, consistent with a unit component sum. Subsequently, P_{ij}^r, L_{ij} and L_{ij}^r are initialized consistently with the neuron values. D^{ria} is initialized in an approximately consistent way for each source node a, with $D^{rba} = 0$.

The following iteration is repeated, until one of the termination criteria (see below) is fulfilled:

Iteration

- For each single request, r, do:
  1. Compute the load L^r_{ij} and subtract from L_{ij} for all arcs (ij).
  2. For each node $i \neq b$, do:
     (a) Calculate E^r_{ij} for each neighbour j, using (5.21).
     (b) Update v^r_{ij} for each neighbour j, using (5.12).
     (c) Update D^{ria} for each source a, using (5.20).
     (d) Update P^r_{ij} for all nodes j, using (5.13).
  3. Compute the new load L^r_{ij} and add to L_{ij} for all arcs (ij).
- Decrease the temperature: $T \to kT$.

The updating process is terminated when the neurons have almost converged to sharp 0/1 states, or if it is obvious that they will not (signaled by a very low T), as defined by the criterion

$$\frac{1}{R(N-1)} \sum_{rij} (v^r_{ij})^2 > 1 - 10^{-6} \;\; \text{AND} \qquad \text{(A.2)}$$
$$\max_{rij} (\Delta v^r_{ij})^2 < 10^{-9} \;\; \text{OR} \;\; T = 0.00001.$$

We have consistently used $k = 0.95$ as the annealing rate. The coefficients α and γ in (5.5, 5.17, 5.21) are chosen as 5 and 0.1 respectively. κ in (5.16, 5.19) are chosen as 10.

For the *single* multicast case, the r-loop disappears; furthermore, the load constraint is irrelevant, so the calculation of loads is unnecessary and the energy should be calulated using the simpler formula in (5.17). For the multiple *unicast* case, the energy should be calculated according to (5.5) and $\mathbf{D}^r$ as (5.11).

Appendix B

Branch-and-Bound Algorithms

As exact solvers for small enough problems, two BB algorithms have been developed, one for single multicasts, the other for multiple problems.

The **single multicast BB** finds a minimal MT, by starting from a tree embryo containing only the sender, and recursively adding paths to each receiver. Each path is recursively constructed, one arc at a time, starting from the receiver end; it is forced to avoid itself, and is complete when the existing partial tree is reached. The lowest MT cost so far is used in every step as a bound to avoid unnecessary searching.[1]

The **multiple multicast BB** is based on an initial recursive generation of the entire set of possible MT's for each request separately. This is done as in the single multicast BB, but no bound is used. For each MT, its cost as well as data on its arc usage are stored. Each MT list is then sorted in increasing cost order.
The proper BB part then consists in a recursive traversal of the space of combinations of one MT for each request, taken from its respective list.

[1]Optionally, an initial cost bound (obtained e.g. from a heuristic) can be specified to further narrow down the search.

Large sets of combinations are avoided before completion, based either on exceeding the lowest total cost found so far (particularly effective due to the sorted lists), or on arc overloading.

The **multiple unicast BB** is contained in the above as an obvious special case, where an MT is a simple path from the sender to the receiver.

Appendix C

The Directed Spanning Tree Heuristic

The single multicast heuristic DSTH consists in the following steps:

1. Find the shortest paths between all pairs of nodes in S (the set of sender and receivers), yielding a distance matrix D restricted to $S \times S$.
2. Interpret the elements of D as arclengths in an auxiliary complete graph G'_s spanning S.
3. Find an approximately minimal directed tree T'_s, spanning G'_s and rooted at the sender node, by starting with a tree containing the sender node only, and repeatedly connecting the node with the shortest arc to the existing tree.
4. Obtain a subnetwork G_s of the original network G by expanding the arcs of T'_s in terms of the corresponding shortest paths in G.
5. Find (as in point 3) an approximately minimal directed tree $T \subseteq G_s$, rooted at the endnode.
6. Finally, obtain a proper MT by removing from T branches not needed to span S.

Bibliography

[1] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, New Jersey (1987).

[2] L. Gislén, C. Peterson and B. Söderberg, "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, 805 (1992).

[3] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling with Potts Neurons", *Neural Computation* **9**, 1627 (1997).

[4] R. Bellman, "On a Routing Problem", *Quarterly of Applied Mathematics* **16**, 87 (1958).

[5] L. Kou, G. Markowsky and L. Berman, " A Fast algorithm for Steiner Trees", *Acta Informatica* **15**, 141 (1981).

[6] J. Häkkinen, M. Lagerholm, C. Peterson and B. Söderberg, "A Potts Neuron Approach to Communication Routing", *LU TP 97-02* (to appear in *Neural Computation*).

[7] R. M. Karp in *Complexity of Computer Computations* Miller and Thatcher, Eds. New York; Plenum Press 85-103 (1972).

[8] M. Lagerholm, C. Peterson and B. Söderberg, "Airline Crew Scheduling Using Potts Mean Field Techniques", *LU TP 97-10.*

[9] J.A. Boyan and M.L. Littman, "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach", in *Advances in Neural Information Processing Systems* **6**, J.D. Cowan,

G. Teauso and J. Alspector, eds., Morgan-Kauffman, San Francisco (1994).

Chapter 6

Clustering ECG Complexes Using Hermite Functions and Self-Organizing Maps

Paper VI

LU TP 98-1
March 30, 1998

Clustering ECG Complexes Using Hermite Functions and Self-Organizing Maps

Martin Lagerholm[1], Carsten Peterson[2]
Complex Systems Group,
Department of Theoretical Physics Lund University,
Sölvegatan 14A, S-223 62 Lund, Sweden

Guido Braccini, Lars Edenbrandt[3]
Department of Clinical Physiology,
Lund University, S-221 85 Lund, Sweden

Leif Sörnmo[4]
Signal Processing Group,
Department of Applied Electronics, Lund University,
Box 118, S-221 00 Lund, Sweden

Submitted to *IEEE Transactions on Biomedical Engineering*

[1] martin.lagerholm@thep.lu.se
[2] carsten.peterson@thep.lu.se
[3] lars.edenbrandt@klinfys.lu.se
[4] leif.sornmo@tde.lth.se

Abstract:

An integrated method for identifying and classifying ECG complexes in the MIT-BIH Arrhythmia Database is presented. The QRS complexes are extracted and the RR intervals are calculated in an autonomous way. Each complex is subsequently decomposed into Hermite basis functions, from which the resulting coefficients and widths are used to represent the beats. By means of this representation, unsupervised self-organizing neural networks are employed to cluster the data into 25 groups. The resulting clusters exhibit a very low degree of misclassification (1.5%). The approach outperforms published supervised learning methods using the same data. The method is also successfully compared with a more conventional template cross-correlation clustering method.

6.1 Introduction

Computer-based methods for analysis and interpretation of ECGs have been subject to intense research for nearly four decades. Electrocardiographs that perform sophisticated signal processing and diagnostic interpretation of signals are today widely spread. Artificial neural networks (**ANN**) is one of the most recent techniques in this field and have primarily been considered for classification of ECGs into different diagnostic groups [1] [2]. It has been shown that ANNs for specific issues can perform better than both experienced cardiologists and ruled-based criteria, e.g. in detecting acute myocardial infarction from the ECG [3]. A first generation of ANNs have also been implemented in commercial electrocardiographs.

Visual analysis of long-term (24h) ECG is tedious and operator dependent. Computer techniques have been developed in order to facilitate the visual analysis, e.g. by condensed print-outs of various signals and trends. With this type of presentation the operator usually can analyze a 24-hour recording in 20-40 minutes provided that no complex arrhythmias exist. It is obvious that automated systems for detection of arrhythmias considerably reduces the amount of time the operator needs to spend. Several commercial systems are available for long-term ECG analysis. However, their performances deteriorate markedly when noise and artefacts are present and, as a consequence, an excessive number of beat classes is created which require considerable manual editing.

Beat classification based on ANN requires that a set of descriptive measurements first has been extracted from the ECG. Several approaches have been taken to this problem of which perhaps the straightforward use of time samples from the ECG signal is most frequently used [4] [5] [6]. Another approach is to use a set of heuristic features e.g. measurements of QRS wave amplitudes and durations [7]. While both these approaches are likely to perform well in recordings acquired during favourable conditions, the classification performance will severely degrade in noisy recordings. A more robust approach to the feature extraction problem is to use a basis function representation of the ECG. This technique was suggested for ECG analysis already in the 60's when Laguerre orthogonal functions were proposed [8]. Later, the Karhunen-Loeve (K-L) expansion, which provides an optimal signal representation in the mean square error sense, was found to be suitable for this purpose, e.g. [9] [10]. It is well-known that the K-L basis functions constitute an orthonormal set and therefore each coefficient in the expansion represents independent information. Based on the shape of the most significant K-L basis functions, the orthogonal Hermite functions were later proposed as a useful parametric model of the QRS complex [11]. An advantage with the Hermite model is that it includes a width parameter. Recently, this model was studied within the context of beat classification [12], and data compression [13].

In this paper, we develop a method for unsupervised characterization of ECG signals. The aim is to partition the beats into clusters that represent central features of the data such that similarity structures between clusters are preserved. Our approach involves Hermite function representation of the QRS complexes in combination with self-organized clustering. In contrast to feedforward supervised ANN learning algorithms, which are trained to identify predefined features in the data, self-organizing networks (**SOM**) organize themselves according to the natural structure of the data – no external teacher is called for [14]. Here, we use a variant of an SOM where neighbours are updated according to a soft-max (Potts) prescription. The input data vector consists of the Hermite coefficients, the width parameter and the lengths of adjacent RR intervals.

Recently, an ANN-based method for beat classification was presented which embraces the "mixture-of-expert" (**MOE**) principle [15]. The present work differs distinctly in several respects to that work, e.g. whereas our approach handles all steps from QRS detection to beat classification, the work in [15] is focused on patient-adaptable techniques that require initial labeling of beats. To the extent it is feasible, our results are compared to those in [15] using the MIT-BIH Arrhythmia database [16].

This paper is organized as follows. In Section 2 we briefly describe the MIT-BIH Arrhythmia database. Our preprocessing techniques, QRS detection and Hermite functions decomposition, are developed in Section 3. Section 4 describes the essentials of the self-organizing map and Section 5 presents the results. The correlation-based method is described in Section 6. Finally, conclusions and outlook can be found in Section 7.

6.2 ECG database

A total of 48 ECG recordings were studied from the MIT-BIH arrhythmia database [16]. Each recording has a duration of 30 minutes and includes two leads, denoted A and B respectively, which are chosen among the modified limb lead II and the modified leads V1, V2, V4, or V5. All beats in the database were annotated by two cardiologists: approximately 70% of the beats were classified as normals while the remaining types of beats were divided into 16 different kinds of abnormal beats. Since we focus on the QRS part of the signal, the "p" beats only containing the P wave are omitted. Hence we only use 15 abnormal beat types (see Table 6.1). When comparing our procedure to the annotations in Table 6.1, we utilize all the 15 different categories rather than lumping them together as normal/abnormal beats.

The ECG signals were resampled from the original 360 Hz sampling rate to 1 kHz in order to be compatible with the software used for high-resolution ECG analysis, including an algorithm for QRS detection [17].

6.3 Preprocessing

The coefficients of the Hermite basis function representation are determined from a highpass filtered ECG signal in order to avoid poor fits due to the presence of very low-frequency components (linear-phase filtering with cut-off frequency at 0.3 Hz). For each QRS complex, the fiducial point is related to the peak location in a signal obtained as the sum of envelopes from the two leads [17]. Next, the ECG signal is extracted from a t_0=200 ms window centered around the fiducial point. The window length is selected on the basis of an exploratory investigation with respect to Hermite function decomposition efficiency and classification

code	beat type	Anno	Our Met	Δ (%)
N	Normal beat	75053	74854	0.3
L	Left bundle branch block	8074	8060	0.2
R	Right bundle branch block	7259	7239	0.3
A	Atrial premature beat	2544	2538	0.2
a	Abberated atrial premature beat	150	109	27.0
J	Nodal (junctional) premature beat	83	83	0.0
S	Supraventricular premature beat	2	2	0.0
V	Ventriculare premature beat	7129	7110	0.3
F	Fusion of ventricular and normal beat	803	798	0.6
b	Ventricular flutter wave	472	458	0.3
e	Atrial escape beat	16	16	0.0
j	Nodal (junctional) escape beat	229	228	0.4
E	Ventricular escape beat	106	106	0.0
P	Paced beat	7028	7014	0.2
f	Fusion of paced and normal beat	982	979	0.3
Q	Unclassifiable beat	33	33	0.0

Table 6.1: The 16 different types of beats from the MIT-BIH Arrhythmia Database with annotations. Also shown is how these are identified with our ECG signal extraction method and the differences (Δ) with the database annotation. The poor detection rate of the a-beats was due to the fact that several very low amplitude PVCs in record 201 where missed by the QRS detector.

performance. The overall efficiency of this extraction procedure when comparing with the database annotation file is high – 99.7%. The 0.3% discrepancy is broken down into beat types in Table 6.1.

Each QRS complex is represented by a $2t_0$-dimensional vector $\vec{x} = \{x_t\}$ which is expanded onto a linear combination of Hermite basis functions $\phi_t^{(n)}(\sigma)$, n=0,1,...,N-1, plus an unknown error term $e_t(\sigma)$ [11] [12],

$$x_t = \sum_{n=0}^{N-1} c_n(\sigma)\phi_t^{(n)}(\sigma) + e_t(\sigma) \tag{6.1}$$

where

$$\phi_t^{(n)}(\sigma) = \frac{1}{\sqrt{\sigma 2^n n! \sqrt{\pi}}} e^{-t^2/2\sigma^2} H_n(t/\sigma) \tag{6.2}$$

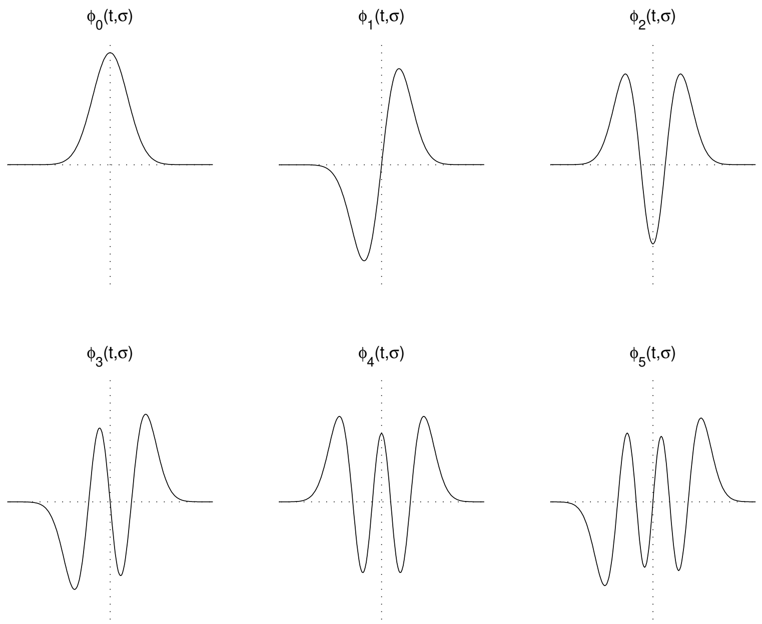

Figure 6.1: The first six Hermite functions (6.2,6.3) as functions of t.

and $H_n(t/\sigma)$ are the Hermite polynomials. The width σ approximates the half-power duration. With $H_0(x) = 1$ and $H_1(x) = 2x$, the Hermite polynomials are recursively given by

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \tag{6.3}$$

Hence each complex is represented by N+1 parameters; σ and $c_n(\sigma)$ (n=0,1,...,N-1). The first six Hermite functions are shown in Fig. 6.1. The Hermite functions form an orthonormal basis

$$\sum_{t=-\infty}^{\infty} \phi_t^{(n)}(\sigma)\phi_t^{(m)}(\sigma) = \delta_{mn} \tag{6.4}$$

for all values of σ, which allows for a cost-effective calculation of $c_n(\sigma)$ in x_t in (6.1).

For a finite window size $\phi_n(\sigma)$ does not obey (6.4). However, if $\phi_n(\sigma)$ is close to zero outside the window it is still a good approximation. In particular, this problem occurs for large n and σ, (see Fig. 6.1); for a given window size, N imposes an upper limit σ_0 on σ. On the other hand, for a good representation of x_t, σ_0 should not be too small. Fortunately, fairly small values of N, i.e. 3 or 4, allow for good beat representation. A window size of $2t_0 = 400$ ms, including a 100 ms zero padding on each side of the QRS complex, is in this respect a suitable choice. Defining $\phi_n(\sigma)$ as being close enough to zero outside the window if

$$|\phi^n_{-t_0}(\sigma)| = |\phi^n_{t_0}(\sigma)| < \frac{1}{10} \max_{t\in[-t_0,t_0]} |\phi_t^{(n)}(\sigma)| \tag{6.5}$$

one obtains for $N = 3$, 4, 5 and 6 the maximum widths $\sigma_0 = 62$, 55, 51 and 47 ms respectively. These numbers are sufficiently large for good representation of the beats.

For a given σ, the coefficients $c_n(\sigma)$ are determined by minimizing the summed square error

$$\sum_t |e_t(\sigma)|^2 = \sum_t |x_t - \sum_n c_n(\sigma)\phi_t^{(n)}(\sigma)|^2. \tag{6.6}$$

The minimum of (6.6), for a fix σ, is conveniently evaluated by means of the orthogonality property (6.4), given by

$$c_n(\sigma) = \vec{x} \cdot \vec{\phi}^{(n)}(\sigma) \tag{6.7}$$

The optimal combination of σ and the coefficients is determined by stepwise incrementation of σ up to σ_0 and then recomputation of (6.7) rather than fitting all parameters simultaneously in order to have a fast decomposition. The use of (6.7) significantly reduces the computational demand when decomposing the signals in Hermite functions. This represents an asset in real-time clustering.

As a relative error measure we use

$$\epsilon = \frac{\vec{e}(\sigma)^2}{\vec{x}^2} \tag{6.8}$$

In general, beats are well represented even with a small number of Hermite functions. This is confirmed both by visual inspection and by the fact that ϵ is low (see Table 6.2). The P and f beat types, which are

Hermite functions (N)	3	4	5	6
$\epsilon(\%)$	9.7	6.8	5.5	4.5
$\langle\sigma\rangle$	19.1	19.0	18.7	18.4

Table 6.2: The relative error ϵ and $\langle\sigma\rangle$ averaged over both leads for N=3, 4, 5 and 6 Hermite functions respectively.

	N	L	R	A	a	J	S	V	F	b	e	j	E	P	f	Q
ϵ (%)	5	2	8	9	7	5	4	5	3	6	3	4	4	14	11	19

Table 6.3: The relative error ϵ for the different beat types when decomposing lead A and B into 5 Hermite basis functions.

pacemaker related, and Q (unclassifiable) are less suitable for representing with a few Hermite functions. However, the variance of ϵ between different types of beats is clearly acceptable when using five Hermite functions are used (see Table 6.3). Figure 6.2 exemplifies the representation of a normal beat with 3, 4, 5 and 6 Hermite functions. As will be shown later (Table 6.5) the performance of the SOM using 4, 5, and 6 Hermite functions representation are approximately the same. We settle for 5 in what follows.

Apart from the above parameters related to the Hermite function representation, local rhythm information will also be used as input to the SOM. The following two features are considered:

$$\begin{aligned} R^- &\equiv R_i^- = R_i^* - R_{i-1}^* \\ R^+ &= \Psi[(R_{i+1}^- - R_i^-) - (R_i^- - R_{i-1}^-)] \end{aligned} \tag{6.9}$$

where R_i^* is the occurrence time for the i:the beat, $\Psi(x)=x$ if $x > 0$, otherwise $\Psi(x)=0$. In this way R^+ does not depend on the heart rate, because it is significantly different from zero only if the i:th beat is premature. This is an important mark for atrial beats.

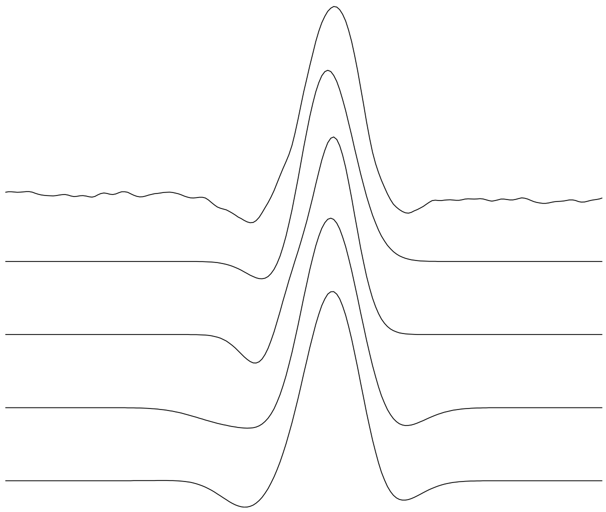

Figure 6.2: An example of Hermite function representation. The upper curve is the original beat (lead A in beat #1 of record 100). The subsequent four curves represent decomposition using 3, 4, 5 and 6 Hermite functions respectively.

6.4 Self-Organizing Map

6.4.1 Self-Organizing Networks

Whereas feedforward supervised ANN learning algorithms are aimed at identifying predefined features in the data, SOMs operate in a different way. The desired feature values are not known beforehand and the network must organize itself according to the natural structure of the data. Hence this approach falls within the family of clustering methods.

Topological Maps

The idea is to map a set of K-dimensional input vectors $\vec{\xi}$ onto a two-dimensional array of output nodes h_j (j=1,...,M), such that the latter capture key features of the input vectors. Each output node correspond to a position in input space given by $\vec{\omega}_j$. The distance between h_j and input vector $\vec{\xi}$ is defined as

$$d_j = \sum_k^K (\omega_{jk} - \xi_k)^2. \tag{6.10}$$

where the index labeling the different input vectors is suppressed. In cases, where each input vector is normalized, this simple measure is sufficient. For each input vector $\vec{\xi}$, a "winner" h_{i^*} is chosen as the one closest to $\vec{\xi}$, i.e.

$$d_{i^*} = \min_j d_j \tag{6.11}$$

Input vectors having h_{i^*} as a winner are said to "belong" to cluster (output node) i^*. In what follows a cluster will also be referred to as a *box*. In the simplest form of self-organization, "winner-takes-all", the weights ω_{mk} are updated such that h_m will become more sensitive to the presented pattern, similar to standard clustering algorithms. This is accomplished by

$$\begin{aligned} \Delta\omega_{mk} &= \eta(\xi_k - \omega_{mk}) \quad && \text{if} \quad m = i^* \\ \Delta\omega_{mk} &= 0 && \text{if} \quad m \neq i^* \end{aligned} \tag{6.12}$$

where η is the step size towards the input vector.

It is often desirable to have a topological order among the output units h_j, such that neighbouring units in the output space will respond to adjacent points in the input space. A commonly used technique to achieve such a *topological map* is by updating units close to the winner node in the same way as the winner node [14] but with a step that in addition to η also depends upon the distance to the winner, r (in the output space). In this way neighbouring units will end up with similar weight vectors. We do this using *Potts* neurons, where the winner "shares" its step with its neighbours. The Potts neuron, $\vec{v}_{i^*j}$, encodes the relative strengths according to

$$v_{i^*m} = \frac{e^{-r_{i^*m}^2/T}}{\sum_k e^{r_{i^*k}^2/T}}. \tag{6.13}$$

The distance r_{ij} (in the output space) between the output units defines the topology of the network. The artificial "temperature" or width T

	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}	ξ_{12}
lead A	c_1	c_2	c_3	c_4	σ							
lead B						c_1	c_2	c_3	c_4	σ		
lead A/B											R^-	R^+

Table 6.4: ξ-representation.

is decreased (annealed) as learning proceeds. The update rule (6.12) is then modified to

$$\Delta\omega_{mk} = \eta(\xi_k - \omega_{mk})v_{i^*m} \text{ for all } m \tag{6.14}$$

Note that

$$\sum_j v_{ij} = 1, \tag{6.15}$$

expressing the "share" philosophy. For $T \to 0$ (6.14) reduces to the "winner-take-all" update rule (6.12). Suitable parameter choices for our procedure are found below.

6.4.2 The QRS Map

Input Structure

We map our preprocessed QRS data, denoted by ξ_k, onto 5×5 maps using (6.10) and (6.11). The ξ_k-representation is shown in Table 6.4 where c_i and σ refer to the Hermite decomposition for each lead. Since the components of the input vector (c_n, σ, R^- and R^+) have different origins one does not *a priori* know their relative importance for distinguishing between various types of beats. Hence we transform the components by subtracting the average and normalizing to a common standard deviation.

We have a mapping from a 12-dimensional input space to a 2-dimensional (5×5) output space. This is in contrast to many applications using SOMs, where the key goal is to reduce the dimensionality of the problem. In our case, that reduction has been achieved with the Hermite function decomposition. One central reason here for using SOMs is to benefit from its topological structure when interpreting the data.

Training

The training, which is divided into two phases, turns out to be robust with regard to all parameters within reasonable windows. In phase I the output vectors evolve with a strong neighbourhood dependence, whereas phase II proceeds with a weaker dependence. The latter allows for outliers to be identified, which are important for diagnosis. Below we describe in more detail the parameters and procedures we use.

Initializing the weights. The input data typically occupies a very small subspace of the total input space. Ideally, one wants the output nodes to focus on this subspace from the start. Since the input vector components have been normalized to zero mean, this is accomplished by initializing ω_{jk} to random[-ε,ε], where ε is chosen small compared to the standard deviation, $\varepsilon = 0.01$.

Local neighbourhood. In order to produce a good topological map, one should start with a wide neighbourhood (T_0 large), compared to the size of the network. This will force all units to become more or less similar. The neighbourhood is then decreased during training, allowing the units to focus on local regions of the input space. For a too small T_0, the coupling between the units is broken and no ordering will emerge. On the other hand, if T is large all the time, all units in the network will end up pointing in the same direction. We decrease the neighbourhood according to

$$T_{t+1} = cT_t \tag{6.16}$$

with $c = 0.95$. The initial T is estimated from the size of the output map as

$$T_0 = -\frac{d^2}{ln(2)} \tag{6.17}$$

where d denotes the diagonal distance of the map. This implies that a output vector at distance d from the winner will be updated with half the strength of the winner.

In phase I, T should not be too low. Hence T is chosen as

$$T = \max(T_{t-1}c, T_{\min}) \tag{6.18}$$

where $T_{\min} = 2/ln(2)$. In phase II there is no lower bound for T.

Learning rate. The learning rate η is in phase I annealed according to

$$\eta_{t+1} = k\eta_t \tag{6.19}$$

with $\eta_0 = 1$ and k = 0.96. In order for the convergence to be reasonable fast, η is chosen as

$$\eta_t = \max(\eta_{t-1}c, \eta_{\min}) \tag{6.20}$$

with $\eta_{\min} = 0.01$. In phase II η is fixed to $\eta_{\min}$.

The weights are updated for every pattern. Other updating rates have been pursued with less efficient training as results.

Termination criteria. The energy E of the system is defined as

$$E(t) = \sum_{ik}(\xi_k^i - \omega_{i^*k}(t))^2 \tag{6.21}$$

where t is the time step and i runs over all the input vectors. The fractional change in E from the last block of 10 time steps to the previous block defines ΔE. Both phases are terminated when $\Delta E < 0.001$.

6.5 Results

6.5.1 Resulting Maps

Depending upon the random number initialization, different features will after completed training settle in different parts of the 5×5 plane. However, the topological relations should be preserved. In Fig. 6.3 we show the beat assignments of the 5×5 map of output nodes, for one of the 30 minute records in the database. Each box represents a cluster of beats. Also indicated is the contents in terms of four kinds of beats; normal beats (N), atrial premature beat (A), ventricular premature beats

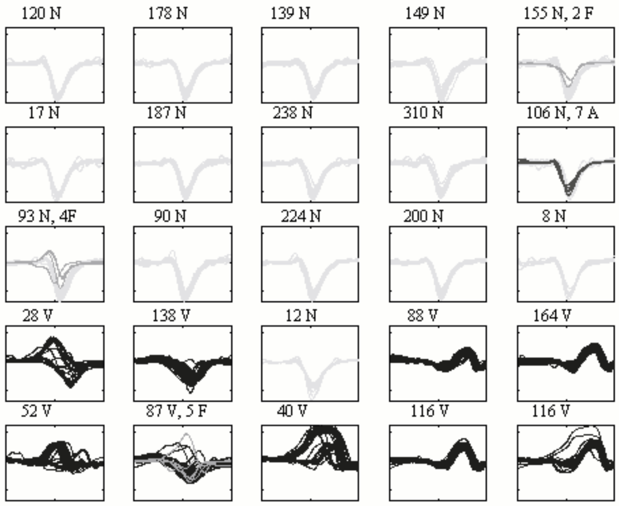

Figure 6.3: Beats from a typical 30 minute record (# 233 lead B) assigned to the different clusters (output nodes). For the notations N, A, V and F see Table 6.1.

(V) and fusion of ventricular and normal beats (F). In this case all 7 A and 11 F beats ends up in clusters where their kind are not dominant. Note that the beats plotted in the boxes are the original beats of lead B, whereas the input to the SOM are composed of the Hermite representation and the R-measures of both leads.

6.5.2 Evaluation of Results

The results above indeed look very good. We next turn to a more quantitative evaluation of the results. First we do this without benchmarking against other unsupervised approaches. Then we compare with the results of [15]. In Section 6 we will make comparisons with a cross-correlation clustering method.

	4 Hermite fcn	5 Hermite fcn	6 Hermite fcn
SOM	1.55 %	1.51 %	1.54 %

Table 6.5: Misclassifications in percentage averaged over all the available beats for different number of Hermite functions.

A beat is defined as misclassified if it ends up in a cluster where the dominant beat is different. The resulting total percentage of misclassified beats for all ECG recordings is equal to 1.5%. The worst misclassification percentage is 12%, which would be obtained if all beats of each record were classified in a single box.

Results for 4, 5 and 6 Hermite functions decomposition are shown in Table 6.5. Table 6.6 shows the result divided into the different beat types. The majority of beats are normal (68.3%) and consequently the likelihood that normal beats will dominate the clusters is high. Most of the normal beats (99.8%) are assigned to clusters in which they represent the dominant fraction. The most common type of misclassification (45.7%) is that non-normal beats are assigned to clusters dominated by the normal ones. Another common type of misclassification is between classes that by definition are very close to each other. The f beats are a mixture between P (paced) and N (normal) beats and the F beats are a mixture between V (ventricular) and N beats. P beats classified as f or vice versa and V beats classified as F or vice versa account for 21.4% of all misclassifications. The unclassifiable beats (Q beats) constitute a very heterogeneous group and, not surprisingly, most of these beats are misclassified.

As mentioned in the introduction, in [15] the MIT-BIH Arrhythmia Data Base [16] was used for classifying beats using ANN methods. Not only do the overall objectives of [16] differ from ours. They also use a different methodology, which does not identify the QRS complexes (the database annotations are used) and, in addition, initial labeling of the beats is required. This is contrast to our approach, which handles all the three steps, QRS extraction, compact representation and classification, in an integrated manner. With regard to the methods used, the representation of each beat is different. Whereas [15] uses the 9 largest principal components computed from 14 samples (180Hz) on each side of the R-peak, we decompose the beats into Hermite functions. Another difference is that we use all 48 records in contrast to [15], where only 33 records (20 for the evaluation), in which premature ventricular contraction can be identified, are kept. Also, the authors of [15] limit their study to a two compressed classes, V and non-V respectively. For the classification, the authors of [15] use a MOE variant of learning vector quantization, whereas in this work we are consistently employing unsupervised learning (clustering). We next compare our results for the classification part with those of [15].

We process the same records as in [15] on a 5×5 map. In order to make comparisons we need to define a proper error measure. In [15] a performance measure is defined as

$$R^{\mathrm{ME}} = \frac{\#\text{ of true negative and true positive beats}}{\text{total }\#\text{ of beats}} \tag{6.22}$$

For comparison we define a similar measure, noting that V in [15] correspond to our V and E,

$$R^{\mathrm{SOM}} = \frac{\text{number of correctly clustered V and E beats}}{\text{total }\#\text{ of beats}} \tag{6.23}$$

The comparison between our SOM and the results from MOE [15] are found in Table 6.7. If an expert were to annotate one typical beat for each of the 25 clusters in our output map while agreeing with the data base annotation, one could consider the beats in that cluster as classified to that type. As can be seen from Table 6.7, within this scenario our method outperforms their method. As a reference a column with the result obtained if all beats where classified as normal (worst case for the SOM approach) is included.

6.6 Crosscorrelation-Based Method

We also gauge our self-organizing approach against a conventional method based on crosscorrelation (**CC**) coefficient and a set of template beats. This method can, similar to the SOM, be categorized as unsupervised classification. Due to memory constraints when implementing the CC method, each of the 48 records in the database is divided into three 10 minutes long records.

The CC method is initiated by using the first beat as a template beat. Each beat is subjected to linear-phase, bandpass filtering in order to remove frequencies which are judged to be less important for the classification procedure (filter cut-off frequencies at 1 and 35 Hz). The subsequent beats are then compared to the template beat with respect to the CC coefficient; each beat is shifted in time until the highest CC coefficient is found. A new template beat is created whenever the CC coefficient drops below a noise-dependent threshold value. The threshold is set to 0.94 at low to moderate noise levels while then gradually decreasing to 0.75 at high noise levels; this threshold design ensures the creation of new beat classes in noisy signals is restricted. The noise level is estimated from the RR interval prior to the QRS complex. A beat classified as being similar to an existing class is then used to update the template beat by means of recursive averaging. This procedure will successively improve the quality of the template beats.

For each 10 minutes record we train a SOM with the same number of clusters ($A \times B$) as produced by the CC method. We compute the total percentage of misclassified beats for both methods (as defined above). The same is done with fixed map sizes of 5×5. The results from the CC method is 4.4% misclassified, which should be compared with 2.3% for the SOM under similar conditions.

6.7 Summary

We have devised a procedure for clustering beats into classes which are not predefined. The method entails Hermite function representation and self-organized neural networks for the purpose of beat clustering. It is found that the QRS complex of the beats in the MIT-BIH Arrhythmia Database are efficiently extracted (99.7%). Decomposing the beats into five Hermite functions turns out to be sufficient for achieving a good classification performance. The entire approach is successfully evaluated in three different and independent ways using 5×5 output maps:

- The degree of misclassification is very low (1.5%), where the former is defined by beats ending up in clusters where the dominant kind is another one. Furthermore, when analyzing the resulting topological maps, these tend to cluster the beats such that similar features are adjacent. The CPU consumption needed for the clustering is modest, less than 1 minute per record on an desktop computer (Digital Alpha 250).
- When comparing the clustering results with those from supervised learning methods [15] by classifying each of the 25 clusters according the dominant kind of beat, our method outperforms the supervised approach.
- As a reference we also classify 10 minutes long records using a method based on template cross-correlation [11]. It turns out that our method performs significantly better than the latter.

One should mention that our approach not only classifies well; it also contains topological information, which is still to be completely exploited. Needless to say, it is very convenient for the cardiologist if similar clusters are presented as neighbours in the map.

It should be noted that our approach embraces all processing steps from the raw ECG signal to the clustered beats; no corrections based on annotations were inserted in between the different steps.

The performance based on an input vector to the SOM which does not include any measurements on signal quality, e.g. information on signal-to-noise ratio. It is reasonable to assume that the performance will become even better when such measurements are included. Another step which remains to be considered is the classification of beat episodes, which would rely on the present classification of single beats combined with information on rhythm.

It is not within the scope of this project to adjust the method to perform particularly well for a certain type of beats. The best performance is found for the normal beats, a class that also outnumbered the other beats. However, if it for medical reasons is considered more important to find for example the ventricular beats it is possible to change the importance of width measures by e.g. simply multiplying that input feature by a factor larger than one.

Acknowledgements: This work was supported by the Swedish National Board for Technical Development and by the Swedish Medical Research Council. We also acknowledge Ola Rauer and Ralf Rittner for contributions at an early stage of this project.

	N	L	R	A	a	J	S	V	F	b	e	j	E	P	f	Q	$\tilde{\Sigma}$
N	74699	0	2	49	1	12	0	49	8	0	0	31	0	0	3	0	155
L	0	8046	0	6	0	0	0	4	0	4	0	0	0	0	0	0	14
R	1	0	7095	139	0	0	0	0	0	4	0	0	0	0	0	0	144
A	281	6	34	2213	0	0	0	4	0	0	0	0	0	0	0	0	325
a	13	0	0	9	57	0	0	30	0	0	0	0	0	0	0	0	52
J	7	0	29	0	0	47	0	0	0	0	0	0	0	0	0	0	36
S	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
V	186	3	0	5	1	0	0	6811	79	23	0	0	1	1	0	0	299
F	146	1	3	0	0	0	0	49	599	0	0	0	0	0	0	0	199
b	0	0	0	4	0	0	0	7	0	446	0	0	1	0	0	0	12
e	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16
j	75	0	6	0	0	0	0	0	0	0	0	147	0	0	0	0	81
E	1	1	22	0	0	0	0	0	0	16	0	0	66	0	0	0	40
P	1	0	0	0	0	0	0	3	0	0	0	0	0	6942	68	0	72
f	15	0	0	0	0	0	0	1	0	0	0	0	0	158	805	0	174
Q	11	2	0	0	0	0	0	2	0	0	0	0	0	5	11	2	31
$\tilde{\Sigma}$	755	13	96	212	2	12	0	149	87	47	0	31	2	164	82	0	

Table 6.6: The SOM clustering of the different types of beats using 5 Hermite functions for the representation and a 5×5 output map. The result from the individual files are added together. The annotation file types are depicted on the vertical axis whereas the clusters dominated by the different types are shown horizontally. For example, 31 of the N beats are assigned to clusters where j is dominant. $\tilde{\Sigma}$ denotes summed rows and columns excluding the diagonal.

record	R^{SOM} (%)	R^{ME}(%)	ref. (%)
200	98.1	81.0	68.0
201	99.9	95.4	89.7
202	99.4	71.7	99.1
203	96.8	87.4	85.1
205	99.9	97.1	97.4
207	96.9	88.3	90.9
208	98.5	90.5	66.3
209	100.0	98.7	100.0
210	98.3	93.4	92.7
213	97.6	91.9	93.2
214	99.9	98.2	88.7
215	99.9	98.4	95.1
219	99.6	97.4	97.0
221	100.0	99.0	83.7
223	99.1	94.4	81.9
228	100.0	99.0	82.3
230	100.0	99.1	100.0
231	99.9	99.8	99.9
233	99.8	97.7	73.0
234	100.0	99.8	99.9
Avg.	99.1	94.0	88.8

Table 6.7: Comparison of the mixture of expert model [15] using the R-measures defined in (6.22) and (6.23). The reference column "ref." shows the extreme case of classifying all beats as normal.

Method	# misclassified
Correlation	4.4%
SOM $A \times B$	2.3%
SOM 5×5	1.3%

Table 6.8: Results of the 10 minutes recording used to compare SOM with the correlation method. The SOM is evaluated for a 5x5 map and for the size produced by the correlation method, $A \times B$.

Bibliography

[1] J.L. Willems, C. Abreu-Lima, P. Arnaud, L.H. van Bemmel, C. Brohet, R. Degani, B. Denis, J. Gehring, I. Graham, G. van Herpen, H. Machado, P.W. Macfarlane, J. Michaelis, S.D. Moulopoulos, P. Rubel, C. Zywietz, "The Diagnostic performance of computer programs for the interpretation of electrocardiograms", *New England Journal of Medicine* **325**, 1767-1773 (1991).

[2] W.G. Baxt, "Application of artificial neural networks to clinical medicine", *Lancet* **346**, 1135-1138 (1995).

[3] B. Hedén, H. Öhlin, R. Rittner, L. Edenbrandt, "Acute myocardial infarction detected in the 12-lead ECG by artificial neural networks", *Circulation* **96**, 1798-1802 (1997).

[4] H.-S. Chow, G.B. Moody, R.G. Mark, "Detection of ventricular ectopic beats using neural networks", *Proc. of Computers in Cardiology, IEEE Computer Society*, 659-662 (1992).

[5] M. Morabito, A. Macerata, A. Taddei, C. Marchesi, "QRS morphological classification using artificial neural networks", *Proc. of Computers in Cardiology, IEEE Computer Society*, 181-184 (1991).

[6] R. Watrous, G. Towell, "A patient-adaptive neural network patient monitoring algorithm", *Proc. of Computers in Cardiology, IEEE Computer Society*, 229-232 (1995).

[7] P. Bozzola, G. Bortolan, C. Combi, F. Pinciroli, C. Brohet, "A hybrid neuro-fuzzy system for ECG classification of myocardial infarc-

tion", *Proc. of Computers in Cardiology, IEEE Computer Society*, 241-244 (1996).

[8] T.Y. Young and W.H. Huggins, "On the representation of electrocardiograms" *IEEE Trans. Biomed. Eng.* **BME-10**, 86-95 (1963).

[9] S. Karlsson, "Representation of ECG records by Karhunen-Loeve expansion", *Dig. 7th Int. Conf. Med. & Biol. Eng., Stockholm*, 105 (1967).

[10] N. Ahmed, P.J. Milne and S.G. Harris, "Electrocardiographic data compression via orthogonal transforms", *IEEE Trans. Biomed. Eng.* **BME-22**, 484-487 (1975).

[11] L. Sörnmo, P.O. Börjesson, M.E. Nygårds and O. Pahlm, "A method for evaluation of QRS Shape features Using a Mathematical Model for the ECG", *IEEE Trans. Biomed. Eng.* **BME-28**, 713-717 (1981).

[12] P. Laguna, R. Jane, P. Caminal, H. Rix and N.V. Thakor, "Adaptive estimation of the QRS complex wave in the electrocardiographic signal (ECG) by the Hermite model: Classification and Ectopic Beat Detection", *Med. Biol. Eng. Comput.* **34**, 58-68 (1995).

[13] R. Jane, S. Olmos, P. Laguna and P. Caminal, "Adaptive Hermite models for ECG data compression: performance and evaluation with automatic wave detection", *Proc. of Computers in Cardiology, IEEE Computer Society*, 389-392 (1993).

[14] T. Kohonen, "Self organized formation of topologically correct feature maps", *Biolological Cybernetics* **43**, 59 (1982); T. Kohonen, *Self-organization and Associative Memory*, third edition, Springer-Verlag, Berlin, Heidelberg (1990).

[15] Y.H. Hu, S. Palreddy and W. J. Tompkins, "A Patient-adaptable ECG beat classifier using a mixture of experts approach", *IEEE Transactions on Biomed. Eng.* **44**, 891-900 (1997).

[16] R. Mark and G. Moody, "MIT-BIH arrhythmia data base directory", Massachusetts Institute of Technology (1988).

[17] M.E. Nygårds, L. Sörnmo, "Delineation of the QRS complex using the envelope of the ECG" *Medical and Biological Engineering & Computing*, **21**, 538-547 (1983).