LU TP 98-5
July 10, 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 Transactions on Communications*

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.

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

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 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 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.

2 Problem Discussion

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.

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.

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 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.

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. 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. 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.

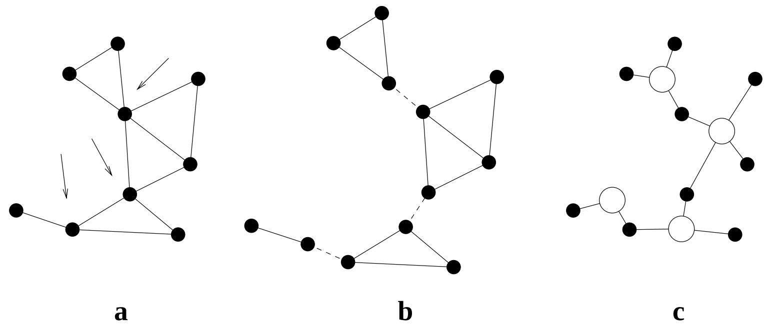

Figure 1: **(a).** A network with the split-nodes marked with arrows. **(b).** The decomposition into subnetworks. **(c).** The hypertree; every second node (white blobs) represents a subnetwork.

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

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

Table 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 $Q \approx 1$.

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 3.1 the Bellman-Ford algorithm for the single unicast

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

problem is recast into a Potts mean field language, while in Subsection 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. 2.

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

$$D_i \to \min_j (d_{ij} + D_j) \equiv \min_j E_{ij},\ i \neq b, \tag{2}$$

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. (2) can be written as

$$D_i = \sum_j v_{ij} E_{ij} \tag{3}$$

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.

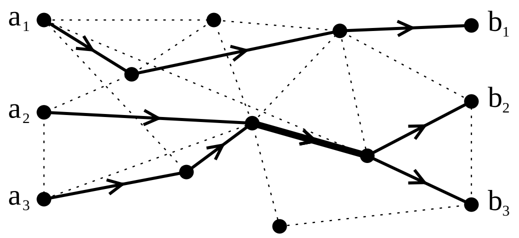

Figure 2: Example of a solution to a 3-request unicast problem. Dotted lines represent unused links, and full lines links that are used by the requests.

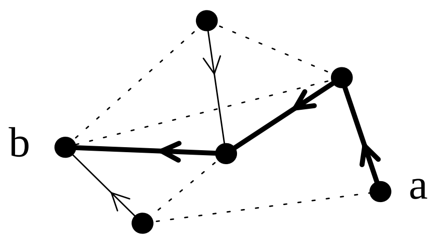

Figure 3: Example of a solution to a single unicast problem. The arcs of the emerging tree are shown as solid lines with arrows indicating the direction, while unused links are shown dotted. The path from **a** to **b** is found by following the arrows; it is marked with fat lines.

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

$$v_{ij} = \frac{e^{-E_{ij}/T}}{\sum_k e^{-E_{ik}/T}} , \tag{4}$$

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 (3, 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, 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. 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.

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^r_{ij} 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

$$E^r_{ij} = d_{ij} + D^r_j + \alpha E^{r,\mathrm{load}}_{ij} + \gamma E^{r,\mathrm{loop}}_{ij} , \tag{5}$$

where D^r_j 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

$$P^r_{ij} = ((\mathbf{1} - \mathbf{v}^r)^{-1})_{ij} = \delta_{ij} + v^r_{ij} + \sum_k v^r_{ik} v^r_{kj} + \sum_{kl} v^r_{ik} v^r_{kl} v^r_{lj} + \dots \tag{6}$$

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^r_i, of how much node i participates in the path $a_r \to b_r$ serving request r,

$$F^r_i \equiv \frac{P^r_{a_r i}}{P^r_{ii}} \frac{P^r_{ib_r}}{P^r_{b_r b_r}} = \frac{P^r_{a_r i}}{P^r_{ii}} \ (\leq 1). \tag{7}$$

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

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

$$W(X) \equiv X\Theta(X) \ , \quad X \equiv L_{ij} - L^r_{ij} - C_{ij} \ , \tag{8}$$

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,

$$E^{r,\mathrm{load}}_{ij} = W(X+1) - W(X) . \tag{9}$$

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

$$E^{r,\mathrm{loop}}_{ij} = \frac{Y}{1-Y} . \tag{10}$$

With a separate Potts system for each request, the updating equations (3, 4) will be replaced by

$$D^r_i \to \sum_j v^r_{ij} E^r_{ij} \tag{11}$$

with

$$v^r_{ij} = \frac{e^{-E^r_{ij}/T}}{\sum_k e^{-E^r_{ik}/T}} \tag{12}$$

The propagators will be updated in a "soft", local manner, relaxing towards (6):

$$P^r_{im} \to \delta_{im} + \sum_j v^r_{ij} P^r_{jm} \ , \quad \text{for all } m. \tag{13}$$

The algorithm defined by iterating (5, 11, 12, 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 $\mathbf{P}^r$.

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 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 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,

$$\Delta_{\mathrm{X}}(\mathrm{Y}) = \frac{D_{\mathrm{Y}} - D_{\mathrm{X}}}{D_{\mathrm{X}}}, \tag{14}$$

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 2.

From Table 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 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.

			legal results					<CPU>		$<\Delta_X(\textsc{pmu})>$		
N	L	R	PMU	BB	IBF	SBF	$\mathrm{T_{BB}}$	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 2: **Multiple unicast**. Number of legal results and CPU time used fro the different algorithms, and a quality comparison. $\mathrm{T_{BB}}$ 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 $\Delta_X(\textsc{pmu})$ indicates that PMU is superior to X.

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.

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. 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

$$D^{ia} = F_{ai} \sum_j v_{ij} \left(d_{ij} + D^{ja}\right), \text{ where } F_{ai} = 1 - \frac{P_{ai}}{P_{ii}} \,. \tag{15}$$

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.

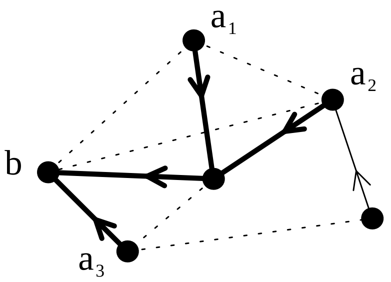

Figure 4: Example of an MT solving a single multicast problem in the transposed network. The nodes a_i, $i = 1, 2, 3$, are receivers and b the sender. The arcs of the emerging BF-tree are shown as solid lines with arrows, with the subset defining the MT marked with fat lines. Unused links are shown dotted.

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

$$\tilde{D}_{ij} = d_{ij} + \frac{\sum_{a \in S} D^{ja} e^{-D^{ja}/(F_{ai}\kappa T)}}{\sum_{a \in S} e^{-D^{ja}/(F_{ai}\kappa T)}} , \tag{16}$$

where κ is a suitably tuned temperature ratio.

The neurons v_{ij}, managing the choice of neighbour, are updated according to (4), with the local energy redefined as

$$E_{ij} = \tilde{D}_{ij} + \gamma E_{ij}^{\text{loop}}. \tag{17}$$

where E_{ij}^{loop} is defined as in (10) but for one request, i.e. neglecting the index r[5]. A load term is not needed; arcs not capable of hosting at least one signal are ignored. Likewise, the propagator is updated in the same manner as for PMU, i.e. (13) neglecting the index r. The resulting algorithm will be referred to as **PSM**.

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 3 shows the number of legal results and the average CPU consumption for the different algorithms, and a performance comparison according to (14 based on the MT-cost. A bar ("—") marks unexplored entries.

From Table 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]Using the loop term will penalize loops immediately; this improves the final result.

			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 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 2; B denotes the number of receivers in each multicast.

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 3.2). Evaluation of the resulting algorithm is done by comparing to BB and a sequential DSTH heuristic.

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,

$$L^r_{ij} = 1 - \prod_{a \in S^r} (1 - \frac{P^r_{ai}}{P^r_{ii}} v^r_{ij}) \ \leq 1 \ . \tag{18}$$

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 (8, 9) with the total arc load given by $L_{ij} = \sum_r L^r_{ij}$. For a loop penalty term, (10) is used without modification. The escape facility is defined and used in the same way as in PMU. In analogy with Subsection 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

$$\tilde{D}^r_{ij} = d_{ij} + \sum_{a \in S_r} D^{rja} \frac{e^{-D^{rja}/(F^r_{ai}\kappa T)}}{\sum_{b \in S_r} e^{-D^{rjb}/\left(F^r_{bi}\kappa T\right)}} \ , \tag{19}$$

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

$$D^{ria} = F^r_{a_r i} \sum_j v^r_{ij} \left(d_{ij} + D^{rja}\right) \ . \tag{20}$$

The local energy corresponding to request r thus takes the form

$$E_{ij}^{r} = \tilde{D}_{ij}^{r} + \alpha E_{ij}^{r,\mathrm{load}} + \gamma E_{ij}^{r,\mathrm{loop}} \ . \tag{21}$$

The neurons and the propagators are updated as in Section 3.2, i.e. using the equations (12) and (13), respectively. The resulting algorithm will be referred to as **PMM**.

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 4.

The results for the different algorithm are shown in Table 4. The PMM heuristic finds approximately the same number of legal results as SDSTH ($\approx$ 1% less), with a consistently better quality.

				legal results				CPU usage		$< \Delta_{\mathrm{X}}(\mathrm{PMM}) >$	
N	L	R	B	BB	SDSTH	PMM	$\mathrm{T_{BB}}$	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 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 2 and 3.

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.

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

$$(\Delta v)^2 = \frac{1}{R(N-1)} \sum_{rij} (\Delta v_{ij}^r)^2 = \frac{1}{R(N-1)} \sum_{rij} (v_{ij}^r(t+1) - v_{ij}^r(t))^2 > 0.1 \tag{A1}$$

the system is reinitialized with $T_0 \to 2T_0$.

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_{ij}^r and subtract from L_{ij} for all arcs (ij).
  2. For each node $i \neq b$, do:
     (a) Calculate E_{ij}^r for each neighbour j, using (21).
     (b) Update v_{ij}^r for each neighbour j, using (12).
     (c) Update D^{ria} for each source a, using (20).
     (d) Update P_{ij}^r for all nodes j, using (13).
  3. Compute the new load L_{ij}^r 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

$$\frac{1}{R(N-1)} \sum_{rij} (v_{ij}^r)^2 > 1 - 10^{-6} \text{ AND } \max_{rij} (\Delta v_{ij}^r)^2 < 10^{-9} \text{ OR } T = 0.00001. \tag{A2}$$

We have consistently used $k = 0.95$ as the annealing rate. The coefficients α and γ in (5, 17, 21) are chosen as 5 and 0.1 respectively. κ in (16, 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 (17). For the multiple *unicast* case, the energy should be calculated according to (5) and $\mathbf{D}^r$ as (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.[6]

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. 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.

References

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

[6]Optionally, an initial cost bound (obtained e.g. from a heuristic) can be specified to further narrow down the search.

[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).