LU TP 95-7
April 1995

A Study of the Mean Field Approach to Knapsack Problems

Mattias Ohlsson[1]

Department of Theoretical Physics, University of Lund
Sölvegatan 14A, S-22362 Lund, Sweden

Hong Pi[2]

Department of Computer Science and Engineering
P.O.Box 91000, Portland, OR 97006-1000, USA

Submitted to **Neural Networks**

Abstract:
The mean field theory (MFT) approach to knapsack problems is extended to multiple knapsacks and generalized assignment problems with Potts mean field equations governing the dynamics. Numerical tests against "state of the art" conventional algorithms shows good performance for the mean field approach. The inherently parallelism of the MFT equations makes them suitable for direct implementations in micro-chips. It is demonstrated numerically that the performance is essentially not affected when only a limited number of bits is used in the mean field equations. Also, a hybrid algorithm with linear programming and mean field components is showed to further improve the performance for the difficult homogeneous NxM knapsack problem.

[1] mattias@thep.lu.se
[2] pihong@cse.ogi.edu

1 Motivation and Results

Suppose you want to go hill climbing carrying a knapsack filled with a selection of possible items. Which items should you choose, as to maximize your comfort, when the knapsack is bounded by volume and weight constraints. This is a simple example of a combinatorial optimization problem with inequality constraints. A more real-world problem is that of resource allocation, such as optimally assigning m tasks to n processors, given the *profit* and the amount of *resource* for each task and the *total resource* available for each processor. These two knapsack problems, where the latter is generally known as the generalized assignment problem, are NP-complete, meaning that no algorithm exists that exactly solves the problem in polynomial time. It is therefore important to develop fast algorithms that find good approximate solutions.

Feed-back artificial neural networks (ANN) have turned out to be powerful in finding good approximative solutions to difficult combinatorial optimization problems [1, 2, 3, 4]. The idea it to map the problem onto binary or M-state neurons (spin variables) with an appropriate energy function. The system is relaxed using mean field theory (MFT) techniques in order to avoid local minima. This procedure, sometimes called mean field annealing (MFA), gives an approximate global minimum of the energy.

In ref. [5] a mean field approach was used for the NxM knapsack problem with encouraging results. In this paper we extend and improve the MFT approach to knapsack problems with the following main results:

- In a VLSI implementation of an ANN, the weights and summations are typically subject to limited precision [6, 7]. We simulate the effects of numerical precision and find that the performance of the ANN-MFT technique can be adequately reproduced on a VLSI micro-chip as long as the weights can be stored with more than 5 bits and summations with more than 7 bits (including one sign bit).

- Linear programming (LP) based on the simplex method is designed for continuous optimization problems, but can be used to find approximate solutions to discrete ones like the knapsack. The LP solutions often consists of a series of 1's and 0's together with a remaining part of real numbers. Using the MFT method for this part and combining the two results increases the performance, compared to the "pure" MFT method, on homogeneous NxM knapsack problems with large M.

- The multiple knapsack and the generalized assignment problem represents two generalizations of the standard knapsack problem. Both problems can be mapped onto M-state Potts neurons with Potts mean field equations for the dynamics. Numerical comparisons against "state of the art" conventional algorithms [8] shows good performance.

2 Review of the Knapsack Problem

2.1 Formulation

The knapsack problem is defined as follows: Given a set of N items and a *knapsack* with

$$
\begin{aligned}
p_j &= \textit{profit} \text{ for item } j \ , \\
w_j &= \textit{weight} \text{ for item } j \ , \qquad (1)\\
c &= \textit{capacity} \text{ of the knapsack} \ . \qquad (2)
\end{aligned}
$$

Pick out a subset of items as to maximize the *utility* U,

$$U = \sum_{j=1}^{N} p_j s_j$$

subject to

$$\sum_{j=1}^{N} w_j s_j \leq c \ , \tag{3}$$

with s_j being $\{0,1\}$ decision variables (item j goes into the knapsack if $s_j = 1$). The variables p_j, w_j and c that define the problem can be either real or integers, but we will assume that they are positive. We will also assume, without loss of generality, that

$$\sum_{j=1}^{N} w_j \ > \ c \tag{4}$$

$$w_j \ \leq \ c \ \forall \ j. \tag{5}$$

If eq. (4) is violated then all $s_j = 1$ and the utility is given by $U = \sum_j p_j$. Furthermore, if eq. (5) does not hold then trivially $s_j = 0$ for each j such that $w_j > c$.

In this "standard" knapsack problem there is only one constraint equation (eq. (3)). One can however think of situations where it is necessary to have two or more equations of constraints. We can imagine the knapsack to be bounded by a weight and a volume constraint. In ref. [5] a more generalized knapsack problem was considered with M equations of constraints. Eqs. (1,2) then reads,

$$
\begin{aligned}
w_j \rightarrow w_{ij} &= \textit{weight} \text{ for item } j \text{ in constraint } i \ , \\
c \rightarrow c_i &= \text{i:th knapsack } \textit{capacity} \ ,
\end{aligned}
$$

and eq. (3) changes to

$$\sum_{j=1}^{N} w_{ij} s_j \leq c_i \ , \ \ (i = 1, ..., M) \ . \tag{6}$$

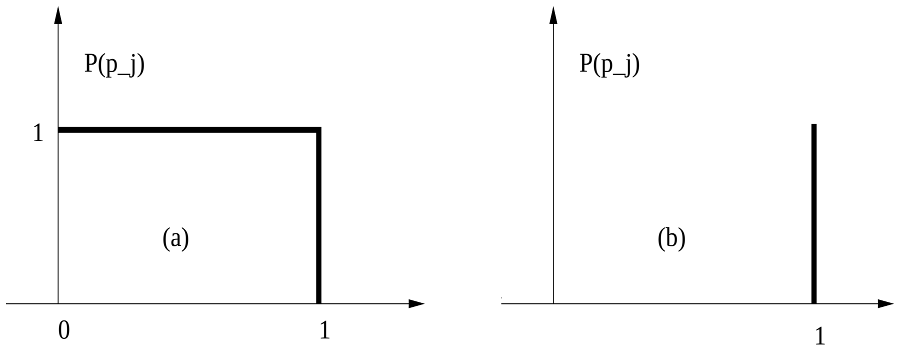

Figure 1: **(a)** Inhomogeneous distributions for the profits. **(b)** All profits are equal to unity (homogeneous) meaning that maximizing the utility is the same as maximizing the number of items in the knapsack.

We will consider a class of problems where w_{ij} and p_j are independent of each other. The weights will be uniform random numbers on the unit interval and the capacities will be fixed to a common value c. For $c \approx N/2$ almost all items will go into the knapsack but it will be nearly empty if $c \ll N/4$. We will use the most difficult case $c = N/4$ where approximately $N/2$ items goes into the knapsack. The profits will be either $p_j \in [0, 1]$ (randomly) or fixed to the value $p_j = 1$ (see fig. 1). These two different distributions for p_j will be referred to as inhomogeneous and homogeneous problems, respectively. For the homogeneous problem maximizing the utility is equivalent to maximize the number of items that goes into the knapsack. This is generally a more difficult problem since many algorithms benefits from a possible ordering of the p_j's.

2.2 The Mean Field Annealing Approach

Since the knapsack problem is NP-complete exact solutions are practically inaccessible for large problem sizes N. It is therefore necessary to develop algorithms that give good approximate solutions in polynomial time. The mean field annealing is one approach. One starts by mapping the problem onto binary neurons with an energy function E, where the constraint equations are implemented in a soft manner using a Lagrangian multiplier. One has [5],

$$E = -\sum_{j=1}^{N} p_j s_j + \alpha \sum_{i=1}^{M} \Phi\left(\sum_{j=1}^{N} w_{ij} s_j - c_i\right) \quad , \tag{7}$$

where $\Phi(x)$ is given by

$$\Phi(x) = \Theta(x) \cdot x \quad , \tag{8}$$

with $\Theta(x)$ being 1 if $x > 0$ and 0 otherwise. The global minimum of E gives us the desired solution. It is however only a reformulation and E is very difficult to minimize.

Local methods such as gradient descent techniques will very easily be trapped in a local minimum. The use of MFT techniques will avoid such local minima and the MFT equations for a system defined by eq. (7) is given by

$$v_i = \frac{1}{2}\left[1 + \tanh\left(-\frac{\partial E}{\partial v_i}\frac{1}{T}\right)\right] \quad (i = 1, ..., N) \ , \tag{9}$$

where the discrete variables s_i have been replaced by the (continuous) mean field variables

$$v_i = < s_i >_T \quad .$$

The "temperature" T has been introduced as an annealing parameter. Start with a high value of T and gradually lower T to a small value while iterating the v_i-equations given by eq. (9).

Due to the non-polynomial form of the constraint term (eqs. (7,8)) special care is needed when calculating the derivative $\partial E/\partial v_i$ appearing in eq. (9). First, one wants to avoid self-coupling terms coming from the non-linear constraint penalties. Second, the derivative is not differentiable at points where $\sum_j w_{ij} s_j = c_i$. We avoid both problems by replacing $\partial E/\partial v_i$ with the difference in E computed at $v_i = 1$ and $v_i = 0$ respectively. One obtains

$$-\frac{\partial E}{\partial v_i} \rightarrow p_i - \alpha \sum_{k=1}^{M}\left[\Phi(\sum_{j=1}^{N} w_{kj} v_j - c_k)|_{v_i=1} - \Phi(\sum_{j=1}^{N} w_{kj} v_j - c_k)|_{v_i=0}\right] \quad .$$

2.3 Hybrid approach - Linear Programming + MFT

Linear programming (sometimes called linear optimization), based on the simplex method [9], is designed to solve continuous optimization problems with both equality and inequality constraints. For the continuous knapsack problem ($s_i \rightarrow x_i \in [0,1]$) one can use LP to get an exact solution in polynomial time (LP scales like N^2M^2). A typical solution for a $N = 30$ and $M = 5$ problem with $p, w \in [0,1]$ randomly and all $c_i = N/4$, looks like

$$\begin{aligned}\vec{x} = \quad & (1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, \\ & 1.00, 1.00, 0.70, 0.15, 0.00, 1.00, 0.00, 1.00, 0.55, 0.00, \\ & 0.00, 0.00, 1.00, 0.00, 0.00, 0.19, 0.00, 0.00, 0.00, 0.00) \ .\end{aligned}$$

One has a series of 1's and 0's and a remaining part of real numbers. The obvious LP algorithm for the *binary* knapsack problem is given by,

$$s_i = \begin{cases} 0 & \text{if } x_i < 1 \\ 1 & \text{otherwise} \end{cases} \ .$$

It is easily augmented with a simple greedy heuristics: For all $s_i = 0$ proceed from larger to smaller p_i and put $s_i = 1$ if it does not violate any constraints. Linear programming + greedy heuristics (LG) finds good solutions, especially for inhomogeneous problems.

Another approach is to combine LP with the mean field method. Given the LP solution with all $x_i = 1$ and $x_i = 0$ fixed, one defines a reduced knapsack problem that only contains items corresponding to the non-integer x_i's. The profits and the weights remains the same but the capacities changes according to

$$c_i \rightarrow c_i' = c_i - \sum_j w_{ij} \delta_{x_j,1} \quad .$$

Having defined this reduced problem one uses the MFT method to find an approximate solution. The combined result from the two methods (called **LM**) is used as the solution for the original problem. This LM method tends to improve the solution quality for homogeneous and large M problems where the LP solution contains a lot of non-integer x_i's.

2.4 Numerical Comparisons

In this section we will compare the performance of the LM method against other algorithms. For small-sized problems ($N, M \leq 30 - 40$) one can check against exact solutions but for large problem sizes comparisons are only possible among the methods for approximate solutions. For all problems the weights are random numbers with $w_{ij} \in [0, 1]$. The capacities c_i are all fixed to the value $N/4$, meaning that approximately half of the items will go into the knapsack. The profits will be either $p_j \in [0, 1]$ randomly or $p_j = 1$ ($j = 1, ..., N$).

Branch and Bound (BB)
For an exploratory search the number of computational steps grows like 2^N. This number of steps can, however, be substantially reduced using a branch and bound tree search [8] checking bounds on constraints and utility for subtrees thereby avoiding unnecessary searching. If there is no (positive) correlation between profits and weights it it likely that an item with large profit will go into the knapsack. Ordering the problem with decreasing profits,

$$p_1 > p_2 > ... > p_N$$

will accelerate the branch and bound technique. For homogeneous problems no such ordering can be done and the BB method may require a large number of computational steps to find the exact solution.

Mean Field (MF)
The decision process is conveniently measured by the saturation $\Sigma = \frac{4}{N} \sum_j (v_i - 0.5)^2$. For small Σ (high T) all the v_i's are close to 0.5 and no decisions are made, but as the v_i's becomes either 1 or 0 Σ increases to finally saturate at $\Sigma = 1$ ($T \rightarrow 0$ limit). The annealing is started at $T_o = 10$ and is lowered according to the schedule

$$T_{n+1} = kT_n \quad , \quad k = \begin{cases} 0.99 & \text{if } 0.1 < \Sigma < \frac{N-1}{N} \\ 0.90 & \text{otherwise} \end{cases} \quad .$$

It is terminated when $\Sigma > 0.999$ and $\Delta < 0.00001$, where $\Delta = \frac{1}{N}\sum_i (v_i(n+1) - v_i(n))^2$ measures the rate of change. We employ a progressive constraint coefficient $\alpha = 1.0/T$ to avoid final constraint violations. A simple greedy heuristics is used if the final solution do violate any constraint. This method scales like NM.

Simulated Annealing (SA)

Simulated annealing [10] is easily implemented in terms of attempted single-spin flips, subject to the constraints. The following update rule will, given a logarithmic annealing schedule, minimize E in eq. (7),

$$s_i = \begin{cases} -s_i & \text{if } \Delta E < 0 \\ -s_i & \text{with Prob}(e^{-\Delta E/T}) \text{ if } \Delta E \geq 0 \end{cases} .$$

For practical reasons we use the faster geometrical decay of the temperature, $T_{n+1} = kT_n$, where $T_o = 15, T_{final} = 0.01$ and $k = 0.995$ in order to get a similar CPU time consumption as the MF algorithm.

Table 1 shows the result for the $N = 30, M = (5, 10, 30)$ problems where the numbers shown on each row are averages over 1000 independent runs. For such a small N it is possible to find the exact solutions using the BB algorithm. As expected the LG (linear programming with greedy heuristics) performs very well for inhomogeneous problems and it is actually slightly better than the MF algorithm. The strength of the LM (LP+MF) approach is apparent for the homogeneous problems where it is a winner.

N	p_j	M	BB	MF	LM	LG	SA
			Utility				
30	$[0,1]$	5	10.49	10.33	10.31	10.39	10.35
		10	10.00	9.82	9.81	9.87	9.86
		30	9.34	9.14	9.15	9.19	9.19
	1	5	16.56	16.29	16.41	16.31	16.02
		10	15.22	14.88	15.01	14.69	14.64
		30	13.57	13.12	13.29	12.61	13.00
			CPU time (sec.)				
30	$[0,1]$	5	0.33	0.04	0.02	0.02	0.10
		10	0.67	0.06	0.04	0.03	0.11
		30	3.4	0.15	0.07	0.05	0.16
	1	5	32.3	0.04	0.04	0.04	0.10
		10	69.4	0.05	0.07	0.06	0.11
		30	319	0.14	0.18	0.13	0.15

Table 1: Comparisons of performance and CPU consumption for different algorithms on small (N=30) knapsack problems. The CPU consumption refers to a DEC Alpha 3000/400 workstation.

In table 2 comparisons are made on large homogeneous ($p_j = 1, j = 1, ..., N$) problems and the numbers are again averages over 1000 runs except for $N = 400$ where only 250 runs were made. The MF and LM algorithms gives consistently higher utilities compared to the LG and SA methods. It is however important to note that the LM scales like N^2M^2 and requires substantially more CPU-time than the faster MF algorithm. The decreasing performance of SA (with increasing N) can be explained by the fact that SA is based on partial phase-space exploration and therefore needs slower annealing schedules with increasing problem size.

		Utility				CPU time (sec.)			
N	M	MF	LM	LG	SA	MF	LM	LG	SA
50	25	23.81	24.18	23.22	23.32	0.22	0.40	0.34	0.40
	50	22.45	22.85	21.57	22.10	0.52	0.81	0.62	0.66
100	50	47.96	48.89	46.88	46.73	1.8	4.5	4.1	2.1
	100	46.03	46.90	44.52	45.09	2.8	6.7	5.8	2.8
200	100	96.24	98.38	94.66	94.10	8.5	37	36	6.9
	200	93.64	95.40	91.22	91.92	20	105	101	17
400	200	193.5	197.4	190.9	189.9	43	447	437	31
	400	190.0	193.1	186.1	187.1	79	1054	1026	57

Table 2: Comparisons of performance and CPU consumption for different algorithms on large knapsack problems with $50 \leq N \leq 400$. The CPU consumption refers to a DEC Alpha 3000/400 workstation.

3 Finite Precision in the MFT Equations

Recurrent networks are notoriously slow when simulated on a serial computer. Massively parallel implementation on a VLSI chip can push the speed to $10^8 - 10^9$ CUPS (connection updates per second) [7]. However, currently the weights and activations on a micro-chip are typically stored with 5 bits (4 bits plus a sign bit), and the summations are typically reproducible to about 10 bits [6, 7]. It is a legitimate question to ask how the MFT performance become degraded when subjected to limited precision.

To measure the effects of finite bits, we use the ratio

$$R = \frac{E(N_v, N_{arg})}{E_0},$$

where E_0 is the utility as given in eq. (7) for a MFT solution with infinite (machine) precision, and $E(N_v, N_{arg})$ is the utility for a MFT solution learned with finite precision. The finite precision effect is simulated by assuming that the decision variable v_i given in eq. (9) is stored by one sign bit plus N_v bits in the range $[0.0, 1.0]$, and the argument of tanh (the summation) in eq. (9) is represented by one sign bit plus N_{arg} bits to cover numbers

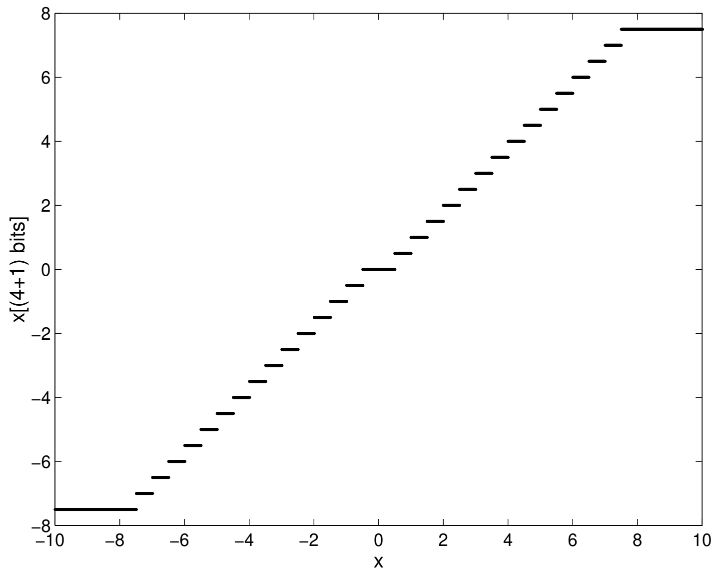

Figure 2: The approximate value of x using 4 bits plus 1 sign bit.

in the range $[-5.0, 5.0]$. We further assume that the chip is able to treat the overflows (underflows) as the maximum (minimum) number represented by the bits. Fig. 2 shows the float numbers as represented by (4+1) bits covering $[-5.0, 5.0]$.

In fig. 3 we show the relative performance R as a function of the finite bits N_v and N_{arg} for a $M = N = 30$ knapsack problem. The numbers in parenthesis are the standard deviation for 10 simulation runs. Fig. 4 shows the same results for a $M = N = 100$ problem. The number of bits in the figures does not include the sign bit. The results show that there is essentially no performance degradation if $N_{arg} \geq 6$ and $N_v \geq 6$. For $N_v = 4$, the number of bits implemented in the Bellcore chip [6] as well as the chip by Murray et al [7], the MFT solution would be adequate although typically it would be 5% off from the optimum.

4 The Generalized Assignment Problem

4.1 Formulation

The generalized assignment problem (GAP) extends the original knapsack problem by introducing more than one knapsack. It can be described as the problem of assigning N tasks to M processors in an optimal way. To be more precise (using the knapsack "language"), given N items and M knapsacks with

$$p_{ij} = \text{profit of item } j \text{ if assigned to knapsack } i$$

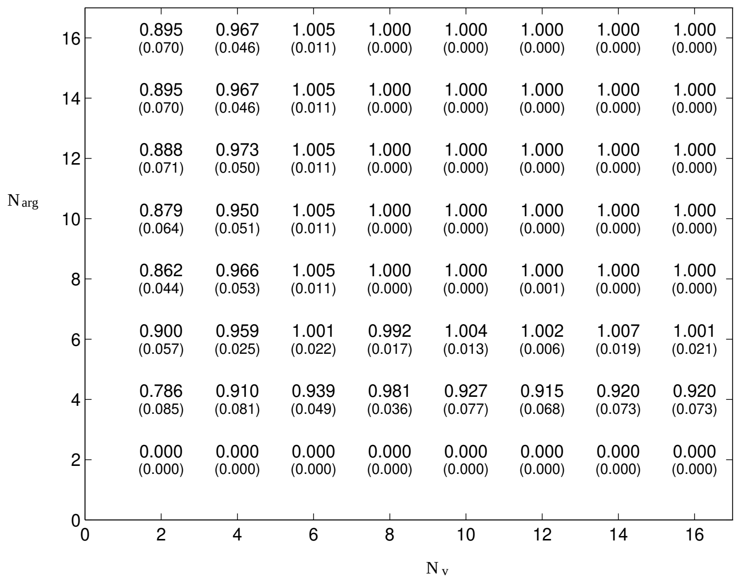

Figure 3: The relative performance R (and its standard deviation in parenthesis) versus the number of bits (not including the sign bit) in v and the argument of tanh in eq. (9) in a 30 by 30 knapsack problem.

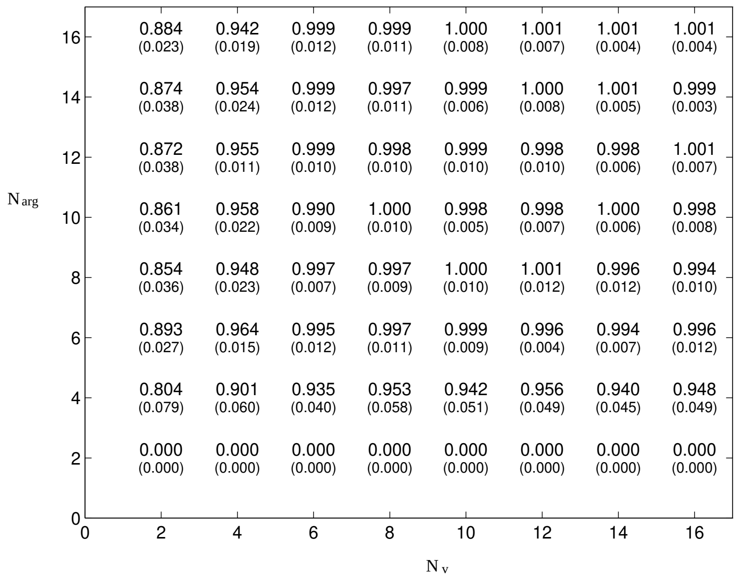

Figure 4: The relative performance R (and its standard deviation in parenthesis) versus the number of bits (not including the sign bit) in v and the argument of tanh in eq. (9) in a 100 by 100 knapsack problem.

$$
\begin{aligned}
w_{ij} &= \text{weight of item } j \text{ if assigned to knapsack } i \\
c_i &= \text{capacity of knapsack } i \ ,
\end{aligned}
$$

assign each item to exactly one knapsack with the objective to maximize the utility U without exceeding the capacity for each knapsack. Formally,

$$
\begin{aligned}
\text{maximize} \qquad & U = \sum_{i=1}^{M} \sum_{j=1}^{N} p_{ij} s_{ij} & (10) \\
\text{subject to} \qquad & \sum_{j=1}^{N} w_{ij} s_{ij} \leq c_i \quad (i = 1, ..., M) & (11) \\
& \sum_{i=1}^{M} s_{ij} = 1 \quad (j = 1, ..., N) & \\
& s_{ij} = 1 \text{ or } 0 \ , &
\end{aligned}
$$

where $s_{ij} = 1$ if item j is assigned to knapsack i and 0 otherwise. There will not always be a feasible solution because of the capacity constraints (eq. (11)). If, however, one allows for possible unassigned items, feasible solutions can always be found. This is called LEGAP.

$$
\begin{aligned}
\text{GAP:} \qquad & \sum_{i=1}^{M} s_{ij} = 1 \quad (j = 1, ..., N) & (12) \\
\text{LEGAP:} \qquad & \sum_{i=1}^{M} s_{ij} \leq 1 \quad (j = 1, ..., N) \ . &
\end{aligned}
$$

$$(13)$$

Any instance of LEGAP can be turned into a GAP by adding an extra knapsack with capacity $c_{M+1} = n$, profits $p_{M+1,j} = 0$ and weights $w_{M+1,j} = 1$, $(j = 1, ..., N)$. This extra knapsack does not contribute to the utility but will always admit a feasible solution. It is however important to note that, given a set of profits, weights and capacities, the GAP and LEGAP solutions will generally not be the same. It may be favorable for LEGAP to allow for possible unassigned items, with small profits and large weights, when maximizing the utility.

If the profits and weights are the same for all knapsacks but may differ from item to item $(p_{ij}, w_{ij} \rightarrow p_j, w_j)$ then LEGAP has turned into a multiple knapsack problem (MKP). MKP will not be treated as a separate problem since it is a special case of LEGAP.

4.2 A Mean Field Approach to GAP

Both GAP and LEGAP are NP-complete and one are stuck to approximate solutions for large N problems. We will again use the MFT equations to construct a polynomial-time algorithm. GAP (eqs. (10-12)) is mapped onto a generic energy function E_G, but with M-state Potts neurons rather than binary one as in the knapsack case. E_G is given by

$$E_G = -\sum_{ij} p_{ij} s_{ij} + \alpha \sum_{i=1}^{M} \Phi\left(\sum_{j=1}^{N} w_{ij} s_{ij} - c_i\right) \quad , \tag{14}$$

where $\vec{s}_j$ is constrained to one of the unit vectors in the M-dimensional hypercube. The MFT equations for $v_{ij} =< s_{ij} >_T$ corresponding to the system defined by eq. (14) is for GAP given by [2],

$$\begin{aligned} v_{ij} &= \frac{e^{u_{ij}}}{\sum_k e^{u_{kj}}} \quad \text{(GAP)} \\ \text{with } u_{ij} &= -\frac{\partial E_G}{\partial v_{ij}} \frac{1}{T} \end{aligned} \tag{15}$$

For the case of LEGAP one also allows for possible "zero"-vectors ($\vec{s}_j = (0, 0, ..., 0)$), which modifies the MFT equations according to

$$v_{ij} = \frac{e^{u_{ij}}}{1 + \sum_k e^{u_{kj}}} \quad \text{(LEGAP)}$$

The derivative $\partial E_G / \partial v_{ij}$ is treated exactly as in the knapsack case. Self-coupling terms are avoided by a linear approximation of $E_G(v_i)$, and one obtains

$$-\frac{\partial E_G}{\partial v_{ij}} \rightarrow p_{ij} - \alpha \left[\Phi(\sum_{k=1}^{N} w_{ik} v_{ik} - c_i)|_{v_{ij}=1} - \Phi(\sum_{k=1}^{N} w_{ik} v_{ik} - c_i)|_{v_{ij}=0} \right] \quad .$$

4.3 Numerical experiments

In this section we will perform some numerical experiments for the generalized assignment problem defined in the previous section (eqs. (10-12)). The profits, weights and capacities are generated according to

$$\begin{aligned} p_{ij}, w_{ij} &\in [1, 100] \quad \text{uniformly random} \\ c_i &= \frac{0.8}{M} \sum_{j=1}^{N} w_{ij} \quad , \quad (i = 1, ..., M) \quad , \end{aligned}$$

where p_{ij}, w_{ij} and c_i are integers. The capacities will generally admit feasible solutions, although they are rather tight. In real world applications one often finds a (positive) correlation between the profits and the weights. In order to simulate such correlations we also consider a second class of problems where the weights are given by, $w_{ij} \in [p_{ij}, p_{ij} + 20]$ (uniformly random).

Mean Field (MF)
For the GAP formulation v_{ij} is given by the Potts update equation (eq. (15)). For large temperatures all v_{ij}'s will merge into a common fix point, $v_{ij} = v_o = 1/M$. The start temperature is taken to be $T_o = 10\max_{(ij)}(p_{ij})$ (≈ 1000) making $v_{ij} \approx v_o$ ($\forall\, i, j$). As in the knapsack case we measure the decision process by the saturation $\Sigma = \frac{1}{N}\sum_{i,j} v_{ij}^2$ which starts at $\Sigma = \frac{1}{M}$ and saturates at $\Sigma = 1$. The annealing schedule is $T_{n+1} = kT_n$ with fixed $k = 0.98$. The constraint coefficient is $\alpha = 25/T$ for all the problems considered here. Final constraint violations are treated by a simple greedy heuristics and may, in some rare cases, lead to unassigned items.

As reference we will use two conventional algorithms, one for exact and one for approximate solutions. The algorithms (including the F77 code) is taken from the reference [8]. The exact method is a depth-first branch and bound (DBB) algorithm and is a generalization of the BB technique for the standard knapsack problem. The algorithm for approximate solutions (MTG) first finds a feasible solution by iteratively assigning items to knapsacks using simple criteria. The current solution is then augmented by local exchanges of items (for details see ref. [8]).

Table 3 shows the result for small N problems where one can compare against the exact solutions. The numbers shown are averages over 1000 independent runs. The MF algorithm performs well for both uncorrelated and correlated weights. It performs significantly better than MTG on problems with small M.

For larger problem sizes exact solutions becomes practically inaccessible and one are stuck to approximate solutions. Table 4 presents the result for experiments with $30 \leq N \leq 200$. The DBB algorithm can be turned into an approximative method by limiting the number of backtrackings performed [8]. The method referred to as DBB1 is limited to 100 backtrackings and DBB2 10 times that amount. It was, however, not possible to use DBB1

		Utility/N			CPU time (sec.)		
N	M	MF	MTG	DBB	MF	MTG	DBB
uncorrelated weights							
20	5	77.5	75.9	78.5	0.15	0.001	2.2
	10	83.8	83.7	85.0	0.41	0.002	10.6
correlated weights							
20	5	39.0	35.3	43.0	0.21	0.001	16.3
	10	35.5	35.6	42.6	0.48	0.002	4.9

Table 3: Numerical comparisons of performance and CPU consumption on GAP's with small N. The mean field (MF) algorithm is checked against an exact depth-first and branch-and-bound (DBB) method and a polynomial-time algorithm (MTG) for approximative solutions. All the test were made on a DEC Alpha 3000/400 workstation.

		Utility/N				CPU time (sec.)			
N	M	MF	MTG	DBB1	DBB2	MF	MTG	DBB1	DBB2
uncorrelated weights									
30	5	78.9	76.7	77.3	78.1	0.21	0.002	0.065	0.48
	10	86.1	85.4	85.5	85.7	0.61	0.003	0.083	0.53
50	5	80.3	77.4	77.9	78.3	0.34	0.003	0.11	0.60
	10	87.7	86.3	86.4	86.4	1.0	0.005	0.16	0.75
100	10	89.0	86.1	86.2	86.2	2.3	0.012	0.56	1.9
	20	93.4	93.1	93.1	93.1	5.8	0.019	0.81	2.9
200	10	89.5	86.0	-	-	5.4	0.024	-	-
	20	94.2	92.1	-	-	14	0.037	-	-
correlated weights									
30	5	41.0	37.1	38.5	40.5	0.28	0.002	0.25	0.76
	10	39.0	37.4	41.3	42.8	0.63	0.004	0.42	1.2
50	5	42.4	38.3	38.6	39.3	0.40	0.003	0.86	1.4
	10	41.8	38.2	40.5	42.6	0.94	0.006	2.8	3.5
100	10	43.6	39.3	39.4	40.4	2.2	0.013	27	28
	20	41.7	40.7	42.0	44.5	5.7	0.022	70	72
200	10	44.0	39.9	-	-	5.1	0.024	-	-
	20	44.1	40.4	-	-	11	0.040	-	-

Table 4: Comparisons for large N GAP's. The DBB1 algorithm is limited to 100 backtrackings while DBB2 allows for 10 times that amount. The CPU consumption refers to a DEC Alpha 3000/400 workstation.

and DBB2 for the $N = 200$ problems because of very long execution time. As usual all numbers are averages over 1000 independent runs. The MF algorithm gives consistently larger utilities compared to the MTG method. The MTG is however a very fast algorithm. Compared to DBB1 and DBB2 the MF algorithm usually finds the better solution and is also faster for the correlated problems.

5 Summary

We have extended the MFT approach to the NxM knapsack problem to include the generalized assignment and multiple knapsack problem. The approach have been successfully tested on various problems, where is scales like NM. A hybrid algorithm that combines linear programming with mean field annealing has been showed to further increase the performance for difficult homogeneous knapsack problems with large M.

Finite precision effects are important when implementing the MFT algorithm on a VLSI chip. It is shown numerically that no performance degradation occurs in the MFT equations when the number of bits N_v used for storing v_i and the number of bits N_{arg} used in the argument for tanh both satisfies $N_{arg}, N_v \geq 6$. For the Bellcore chip [6], where $N_v = 4$, the performance is approximately 5% off from the optimum.

Acknowledgements

We would like to thank Prof. Carsten Peterson for valuable discussions during the course of this work. This paper was in part supported by Ellemtel AB. The second author also gratefully acknowledge funding provided by the Advanced Research Projects Agency and the Office of Naval Research under grant ONR N00014-92-J-4062.

References

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

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

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

[4] L. Gislén, C. Peterson and B. Söderberg "Complex Scheduling with Potts Neural Networks", *Neural Computation* **4**, (1992) 805-831.

[5] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints - the Knapsack Problem", *Neural Computation* **5**, (1993) 331-339.

[6] J. Alspector, R.B. Allen, A. Jayakumar, T. Zeppenfeld and R. Meir, "Relaxation Networks for Large Supervised Learning Problems", *Advances in Neural Information Processing Systems 3*, (1991) 1015-1021.

[7] M. Murray, J.B. Burr, D.G. Stork, M.T. Leung, K. Boonyanit, G.J. Wolff and A.M. Peterson, "Deterministic Boltzmann Learning VLSI", *Proceedings of the International Conference on Application-Specific Array Processors ASAP-92*, (1992) 206-217.

[8] S. Martello and P. Toth, *Knapsack Problems - Algorithms and Computer Implementations*, John Wiley & Sons, Chichester (1990).

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