Solving the Assignment Problem with the Improved Dual Neural Network*

Xiaolin Hu[1] and Jun Wang[2]

[1] State Key Laboratory of Intelligent Technology and Systems,
Tsinghua National Laboratory for Information Science and Technology (TNList),
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, China
[2] Department of Mechanical and Automation Engineering,
The Chinese University of Hong Kong, Shatin,
N.T., Hong Kong, China

Abstract. During the last two decades, several neural networks have been proposed for solving the assignment problem, and most of them either consist of $O(n^2)$ neurons (processing units) or contain some time varying parameters. In the paper, based on the improved dual neural network proposed recently, we present a new assignment network with 2n neurons and some constant parameters only. Compared with the existing neural networks for solving the assignment problem, its more favorable for implementation. Numerical simulation results indicate that the time complexity of the network is $O(n)$.

Keywords: Assignment problem, sorting problem, quadratic programming, linear programming, analog circuits.

1 Introduction

The assignment problem is concerned with assigning n entities to n slots for achieving minimum cost or maximum profit. It is known to be a polynomial combinatorial optimization problem. Its applications cover pattern classification, machine learning, operations research and so on.

For solving the assignment problem, there exist many efficient iterative algorithms such as the auction methods [1], signature methods [2]. Inspired by the Hopfield network for solving optimization problems with neural networks, many continuous methods were developed for solving the assignment problem (e.g., [3–7]). One of the major advantages of this kind of methods is that they can be

* X. Hu's work was supported by the National Natural Science Foundation of China under Grant Nos. 60805023 and 90820305, National Basic Research Program (973 Program) of China under Grant no. 2007CB311003, and Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList). J. Wang's work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Grants CUHK417209E and CUHK417608E.

D. Liu et al. (Eds.): ISNN 2011, Part I, LNCS 6675, pp. 547–556, 2011.
© Springer-Verlag Berlin Heidelberg 2011

implemented in parallel analog circuits to achieve super high speed, which is very attractive in real-time applications. However, most of these methods requires an "annealing" procedure which is sensitive to the solution quality. In addition, this time-varying procedure would pose difficulties in circuits implementation.

In [8], an assignment neural network is proposed without any time-varying procedure. It features elegant theoretical results with constant parameters. The major demerit lies in its complex structure. For instance, it entails more neurons and interconnections than the network in [7]. But this is a tradeoff between the efficiency and structural complexity, as argued in [8]. In the present paper, we show that this tradeoff is unnecessary. Based on the *improved dual neural network* or IDNN proposed in [9], we have designed a very simple assignment network with constant parameters only.

2 Problem Formulation

Suppose that there are n entities to be assigned to n slots and assigning entity i to slot j induces cost c_{ij}. Then, what is the best assignment in terms of minimum total cost? This is called *assignment problem*. Mathematically, it can be formulated as a zero-one programming problem:

$$\begin{array}{rl} \text{minimize} & f_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{subject to} & \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n \end{array} \tag{1}$$

where c_{ij} and x_{ij} are, respectively, cost variable and decision variable associated with assigning entity i to slot j. The variable $x_{ij} = 1$ means assigning entity i to slot j and $x_{ij} = 0$ means not assigning entity i to slot j. Since any entity should be, and must be, assigned to only one slot, and any slot should be, and must be, assigned one entity, a feasible assignment should correspond to a matrix $\mathbf{x} = \{x_{ij}\}$ with only one element equal to one in every column and row.

Lemma 1. *The problem* (1) *is equivalent to the following problem*

$$\begin{array}{rl} \textit{minimize} & f_2 = \frac{q}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \textit{subject to} & \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n, \end{array} \tag{2}$$

where $q > 0$ is a constant.

Proof. This can be proved by showing that $\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2$ is a constant in the feasible region. Because $x_{ij} \in \{0, 1\}$, $x_{ij} = x_{ij}^2$. Then

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} = n,$$

which completes the proof.

Problem (1) and problem (2) are zero-one programming problems. But, if the problem has a unique solution, it is known that (1) is equivalent to the following (continuous) linear programming problem

$$\begin{array}{rl} \text{minimize} & f_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} & \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1,\ldots,n \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1,\ldots,n \\ & x_{ij} \in [0,1] \quad \forall i,j = 1,\ldots,n. \end{array} \tag{3}$$

In what follows, we formulate the continuous counterpart of problem (2).

Theorem 1. *If the problem* (2) *has a unique solution, then there exists a sufficiently small positive constant* q *such that* (2) *is equivalent to the following problem*

$$\begin{array}{rl} \textit{minimize} & f_2 = \frac{q}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \textit{subject to} & \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1,\ldots,n \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1,\ldots,n \\ & x_{ij} \in [0,1] \quad \forall i,j = 1,\ldots,n. \end{array} \tag{4}$$

Proof. What is only needed to show is that the unique solution of (2), denoted by $\mathbf{x}^*$, is also a solution of (4), by considering that the objective function of (4) is strictly convex and it has at most one solution.

Denote the feasible region of (4) by $\mathcal{X}$ and the feasible region of (2) by $\mathcal{V}$. Note that any point in $\mathcal{X}$ is called a *doubly stochastic matrix* and any point in $\mathcal{V}$ is called a *permutation matrix*. According to the well-known Birkhoff-von Neumann theorem, a square matrix is doubly stochastic if and only if it is a convex combination of permutation matrices. Namely, any $\mathbf{x} \in \mathcal{X}$ can be expressed as

$$\mathbf{x} = \sum_{k} \theta_k \mathbf{v}^{(k)},$$

where $\mathbf{v}^{(k)} \in \mathcal{V}$, $\sum_k \theta_k = 1, 0 \le \theta_k \le 1$. Denote the unique solution of (2) by $\mathbf{x}^*$. Then

$$\begin{aligned} \langle \mathbf{x}^*, \mathbf{x}^* - \mathbf{v}^{(k)} \rangle &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^*(x_{ij}^* - v_{ij}^{(k)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} (x_{ij}^*)^2 - x_{ij}^* v_{ij}^{(k)} \\ &= n - \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^* v_{ij}^{(k)} > 0 \end{aligned}$$

for $\mathbf{v}^{(k)} \in \mathcal{V}, \mathbf{v}^{(k)} \neq \mathbf{x}^*$, where $\langle \cdot, \cdot \rangle$ stands for the Frobenius inner product of two matrices. Because $\mathbf{x}^*$ is also the unique solution of (3), according to the equivalence between convex optimization problem and variational inequality [10], we have

$$\langle \nabla f_1(\mathbf{x}^*), \mathbf{x} - \mathbf{x}^* \rangle = \langle \mathbf{c}, \mathbf{x} - \mathbf{x}^* \rangle \ge 0 \quad \forall \mathbf{x} \in \mathcal{X}.$$

Now we show that for any $\mathbf{x} \in \mathcal{X}$ but $\mathbf{x} \neq \mathbf{x}^*$, the above equality cannot hold. Otherwise, there exists such a point $\bar{\mathbf{x}}$ so that $\langle \mathbf{c}, \bar{\mathbf{x}} - \mathbf{x}^* \rangle = 0$. Then

$\langle \mathbf{c}, \mathbf{x} - \mathbf{x}^* \rangle = \langle \mathbf{c}, \mathbf{x} - \bar{\mathbf{x}} \rangle \geq 0, \forall \mathbf{x} \in \mathcal{X}$, indicating that $\bar{\mathbf{x}} \neq \mathbf{x}^*$ is also a solution of problem (3), which contradicts to the uniqueness of the solution. Therefore

$$\langle \mathbf{c}, \mathbf{x} - \mathbf{x}^* \rangle > 0 \quad \forall \mathbf{x} \in \mathcal{X}, \mathbf{x} \neq \mathbf{x}^*.$$

Consequently,

$$\langle \mathbf{c}, \mathbf{v}^{(k)} - \mathbf{x}^* \rangle > 0 \quad \forall \mathbf{v}^{(k)} \in \mathcal{V}, \mathbf{v}^{(k)} \neq \mathbf{x}^*,$$

as $\mathcal{V} \subset \mathcal{X}$. Let

$$0 < q \leq \frac{\langle \mathbf{c}, \mathbf{v}^{(k)} - \mathbf{x}^* \rangle}{\langle \mathbf{x}^*, \mathbf{x}^* - \mathbf{v}^{(k)} \rangle}, \quad \forall \mathbf{v}^{(k)} \in \mathcal{V}, \mathbf{v}^{(k)} \neq \mathbf{x}^*.$$

It follows that

$$\langle \nabla f_2(\mathbf{x}^*), \mathbf{v}^{(k)} - \mathbf{x}^* \rangle = \langle q\mathbf{x}^* + \mathbf{c}, \mathbf{v}^{(k)} - \mathbf{x}^* \rangle \geq 0, \quad \forall \mathbf{v}^{(k)} \in \mathcal{V}, \mathbf{v}^{(k)} \neq \mathbf{x}^*.$$

For any $\mathbf{x} \in \mathcal{X}$,

$$\langle \nabla f_2(\mathbf{x}^*), \mathbf{x} - \mathbf{x}^* \rangle = \langle \nabla f_2(\mathbf{x}^*), \sum_k \theta_k \mathbf{v}^{(k)} - \mathbf{x}^* \rangle = \sum_k \theta_k \langle \nabla f_2(\mathbf{x}^*), \mathbf{v}^{(k)} - \mathbf{x}^* \rangle \geq 0,$$

where $0 \leq \theta_k \leq 1, \sum_k \theta_k = 1$. Hence, $\mathbf{x}^*$ is a solution of (4), which completes the proof.

From the proof, it can be seen that q should be small enough and its upper bound is

$$\min_k \frac{\langle \mathbf{c}, \mathbf{v}^{(k)} - \mathbf{x}^* \rangle}{\langle \mathbf{x}^*, \mathbf{x}^* - \mathbf{v}^{(k)} \rangle} \tag{5}$$

which is positive. In practice, the optimal solution $\mathbf{x}^*$ is unknown, and it is time consuming to find all feasible solutions $\mathbf{v}^{(k)}$ to problem (1) or (2). So, it is suggested to set q to be very small. Throughout the rest of the paper, it is assumed that the solution of the assignment problem (1) is unique. Then, with small q, all of the four problems (1), (2), (3), (4) are equivalent.

3 The Improved Dual Neural Network

3.1 Architecture

The problem (4) is a quadratic programming problem. According to [9], it can be solved by using the improved dual neural network (IDNN):

– state equations

$$\begin{cases} \tau \dfrac{du_i}{dt} = -\displaystyle\sum_{j=1}^{n} g(u_i + v_j - c_{ij}/q) + 1, \quad i = 1, \ldots, n \\ \tau \dfrac{dv_i}{dt} = -\displaystyle\sum_{j=1}^{n} g(u_j + v_i - c_{ji}/q) + 1, \quad i = 1, \ldots, n \end{cases} \tag{6a}$$

– output equations

$$x_{ij} = g(u_i + v_j - c_{ij}), \quad i, j = 1, \ldots, n, \tag{6b}$$

where $\tau > 0$ is the time constant and

$$g(s) = \begin{cases} 0, & s < 0, \\ s, & 0 \le s \le 1, \\ 1, & s > 1. \end{cases}$$

Clearly, this network would entail n neurons for representing u_i and n neurons for representing v_i. The block diagram of the neuron u_i is plotted in Fig. 1. Neurons corresponding to the dual variables v_i can be constructed similarly.

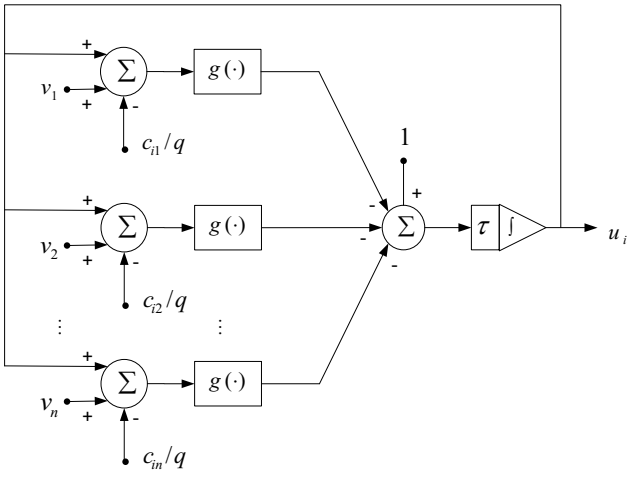


Fig. 1. Block diagram for realizing neuron u_i

3.2 Comparison with Other Neural Networks

In [7], two neural networks were proposed for solving the assignment problem (1). One is called the *primal neural network*, whose dynamic equations are

– state equations

$$\tau \frac{du_{ij}}{dt} = -\sum_{k=1}^{n}(x_{ik} + x_{kj}) + 2 - \alpha c_{ij} \exp\left(-\frac{t}{T}\right), \quad i, j = 1, \ldots, n \tag{7a}$$

– output equations

$$x_{ij} = h(u_{ij}), \quad i, j = 1, \ldots, n, \tag{7b}$$

where $\tau > 0$ is the time constant, $\alpha > 0$ and $T > 0$ are scaling constants, and h is a nonnegative and monotone nondecreasing activation function satisfying

$g(z) \geq 0$ and $dg/dz \geq 0$ for any $z \in \mathcal{R}$ (a typical example is the sigmoid function). The other model is called the *dual neural network*, whose dynamic equations are

– state equations

$$\begin{cases} \tau \dfrac{du_i}{dt} = -\displaystyle\sum_{j=1}^{n} \phi(u_i + v_j - c_{ij}) + \alpha \exp\left(-\frac{t}{T}\right), \quad i = 1, \ldots, n \\ \tau \dfrac{dv_i}{dt} = -\displaystyle\sum_{j=1}^{n} \phi(u_j + v_i - c_{ji}) + \alpha \exp\left(-\frac{t}{T}\right), \quad i = 1, \ldots, n \end{cases} \tag{8a}$$

– output equations

$$x_{ij} = \delta(u_i + v_j - c_{ij}), \quad i, j = 1, \ldots, n, \tag{8b}$$

where τ, α, T are positive constant, ϕ is a nonnegative and nondecreasing function defined as $\phi(z) = 0$ if $z \leq 0$ and $\phi(z) > 0$ if $z > 0$, and δ is a function defined as $\delta(z) = 1$ if $z = 0$, or $\delta(z) = 0$ otherwise. It is seen that the primal neural network entails n^2 neurons and $O(n^2)$ connections whereas the dual neural network entails $2n$ neurons and $O(n^2)$ connections (see [7] for details). The latter model is simpler in structure. In addition, from (8), it can be seen that the dual neural network is nearly as simple as the IDNN (6).

The two models above were derived from the *deterministic annealing neural network* for solving the general convex optimization problems [11]. So, both of them involve a temperature parameter $\alpha \exp(-t/T)$, which is hard to be realized in hardware.

In [8], a neural network called *primal-dual neural network* was proposed for solving the assignment problem. The state equations are

$$\begin{cases} \tau \dfrac{dx_{ij}}{dt} = -\left(c_{ij} \displaystyle\sum_{p=1}^{n}\sum_{q=1}^{n}(c_{pq}x_{pq} - u_p - v_p) + \phi(-x_{ij}) + \sum_{l=1}^{n}(x_{il} + x_{lj}) - 2 \right), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i, j = 1, \ldots, n \\ \tau \dfrac{du_i}{dt} = -\left(-\displaystyle\sum_{p=1}^{n}\sum_{q=1}^{n}(c_{pq}x_{pq} - u_p - v_p) + \sum_{l=1}^{n}\phi(u_i + v_l - c_{il}) \right), \ i = 1, \ldots, n \\ \tau \dfrac{dv_i}{dt} = -\left(-\displaystyle\sum_{p=1}^{n}\sum_{q=1}^{n}(c_{pq}x_{pq} - u_p - v_p) + \sum_{l=1}^{n}\phi(u_l + v_i - c_{li}) \right), \ i = 1, \ldots, n \end{cases} \tag{9}$$

where $\tau > 0$ and ϕ is defined the same as in (8). This network entails constant parameters only, which is superior to the networks (7) and (8). However, it is more complicated in architecture because it consists of $n^2 + 2n$ neurons and $O(n^2)$ connections.

Since the assignment problem can be equivalently written as a linear programming problem or quadratic programming problem, all linear programming

networks and quadratic programming networks are capable of solving it. Currently, there are two main streams of researches in this area, characterized by continuous or discontinuous activation functions. Some typical networks in the former refer to [12–14]. However, it is easy to see that all of them require n^2+2n neurons for solving the assignment problem, though most of them are relatively simpler than the primal-dual network (9). Some typical networks in the latter refer to [15, 16]. It is easy to see that they require at least n^2 neurons for this problem.

3.3 Stability and Convergence

The following result follows from Theorem 2 in [9] and Theorem 1, directly.

Lemma 2. *Let* (u_i^*, v_i^*) *be the equilibrium point of* (6)*, then the unique solution of the assignment problem* (1) x_{ij}^* *can be denoted by* $g(u_i^* + v_j^* - c_{ij})$.

Since it is assumed that the problem (4) has a unique solution, from Theorem 4 in [9] and Lemma 2, it follows the following theorem.

Theorem 2. *If* q *is sufficiently small, any equilibrium point of the network* (6) *is stable in the sense of Lyapunov, and the corresponding output trajectory globally converges to the unique solution of the assignment problem* (1).

4 Numerical Simulations

To verify the correctness of the results presented in last section, we have numerically simulated the proposed network (6) for solving some problems in MATLAB with the "ode23" function.

We randomly generated some cost matrices $\mathbf{c}$ with every element between zero and one. Fig. 2 shows the state trajectories of the network (6) for solving such a problem with $n = 10$. The parameters were set as follows: $\tau = 10^{-6}$ and $q = 0.001$. The output of the network converged to the correct solution of (1), verified by solving the linear programming problem with the MATLAB build-in function "linprog".

According to Theorem 1, q should be set sufficiently small. But it is unclear how small is enough and what is the influence of the values of q on convergence properties. We investigated this issue numerically. The idea is to compare the convergence time of the network with different q values. Let $\mathbf{x}^*$ be the *ground truth* obtained by MATLAB function "linprog". For saving CPU time in running simulations, we terminated the program when

$$\sum_i \sum_j |\text{round}(x_{ij}(t)) - x_{ij}^*| \leq 10^{-6}$$

where round(x) is equal to 0 if $x < 0.5$ and 1 otherwise. The time at which this was achieved was recorded as the convergence time. For every q value, 30 different runs with random initial points between $[-50, 50]$ were executed. The

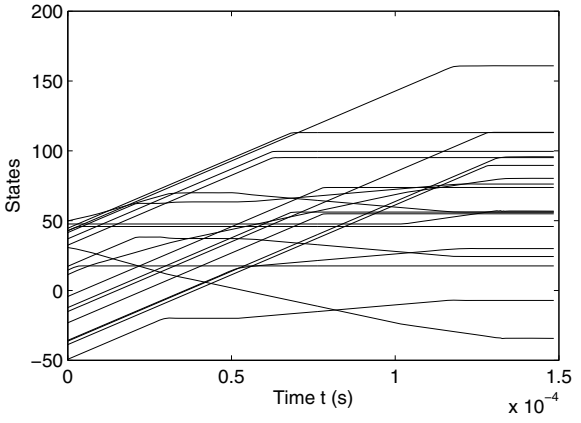
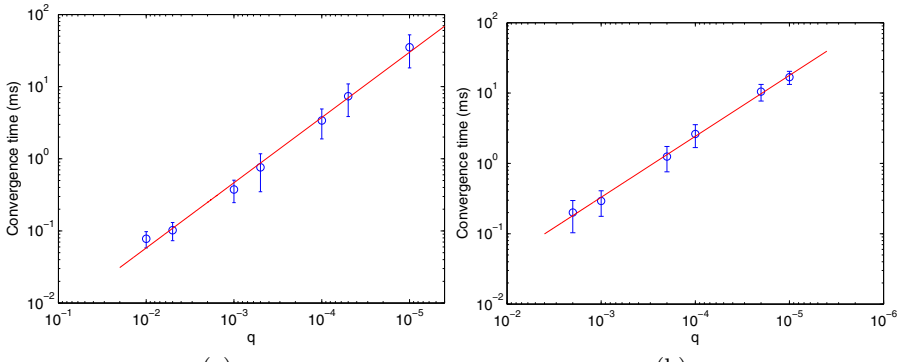


Fig. 2. State trajectories of the network (6)



(a) (b)

Fig. 3. Convergence time of the network (6) with different q values for (a) $n = 10$ and (b) $n = 50$. Note that the coordinates are in logarithm scale. The circles denote the mean and the bars below and above them stand for one standard deviation each. The continuous line in each plot is the linear regression of the logarithm of the mean versus the logarithm of q.

statistics of the convergence time for two problem sizes is plotted in Fig. 3. It is seen that as q decreases the convergence time increases, and the logarithm of the convergence time is roughly a linear function of the logarithm of q. In other words, the convergence time increases as q decreases. Therefore, q should not be too small. This poses some difficulty in choosing an appropriate q.

Next, we investigated the relationship between the problem size and the convergence time of the network. Let $q = 0.01/n$. For each $n = 10, 20, 30, 40, 50, 60, 70, 80, 100$, thirty different runs with random initial points between [-50,50] were executed. The same stopping criterion as above was adopted. The statistics of

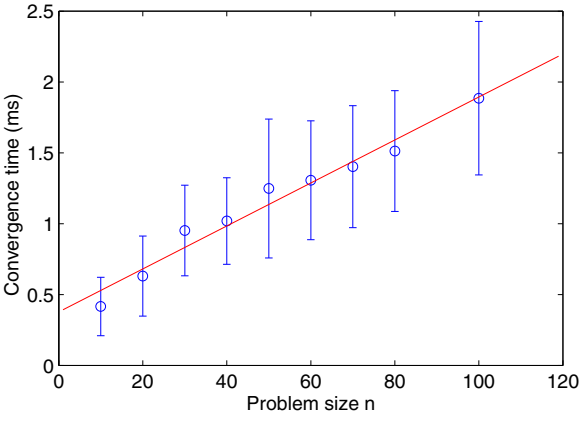


Fig. 4. Convergence time of the network (6) for different problem sizes. The circles and bars denote the mean and the standard deviation, respectively. The continuous line is the linear fit of the means versus n.

the convergence time is plotted in Fig. 4. It is seen that the convergence time grows linearly with the problem size, which is in sharp contrast with the iterative algorithms such as [2] which has $O(n^3)$ time complexity.

5 Concluding Remarks

We applied the *improved dual neural network* or IDNN for solving the assignment problem. An assignment network with $2n$ neurons, free of time-varying parameters, was obtained, which was theoretically guaranteed to be globally convergent to the solution of the problem if only the solution is unique. Numerical simulations indicated that the computing time was roughly a linear function of the problem size n, implying that this network method, if implemented in hardware, would be much faster than traditional iterative algorithms for solving large scale problems.

The main disadvantage of the new assignment network is that it introduces a parameter that should be set appropriately. If it is too large, the network may converge to incorrect states; if it is too small, the convergence will be slow. In most cases, it has to be selected by trial and error.

References

1. Bertsekas, D.P.: A new algorithm for the assignment problem. Math. Prog. 21(1), 152–171 (1981)
2. Balinski: Signature methods for the assignment problem. Operations Research 33(3), 527–536 (1985)
3. Eberhardt, S.P., Daud, T., Kerns, D.A., Brown, T.X., Thakoor, A.P.: Competitive neural architecture for hardware solution to the assignment problem. Neural Networks 4, 431–442 (1991)

4. Wang, J.: Analogue neural network for solving the assignment problem. Electronics Letters 28(11), 1047–1050 (1992)
5. Kosowsky, J.J., Yuille, A.L.: The invisible hand algorithm: solving the assignment problem with statistical physics. Neural Networks 7(3), 477–490 (1994)
6. Urahama, K.: Analog circuit for solving assignment problem. IEEE Trans. Circuits Syst. I 40, 426–429 (1994)
7. Wang, J.: Primal and dual assignment networks. IEEE Trans. Neural Netw. 8(3), 784–790 (1997)
8. Wang, J., Xia, Y.: Analysis and design of primal-dual assignment networks. IEEE Trans. Neural Netw. 9(1), 183–194 (1998)
9. Hu, X., Wang, J.: An improved dual neural network for solving a class of quadratic programming problems and its k-winners-take-all application. IEEE Trans. Neural Netw. 19(12), 2022–2031 (2008)
10. Kinderlehrer, D., Stampcchia, G.: An Introduction to Variational Inequalities and Their Applications. Academic, New York (1980)
11. Wang, J.: A deterministic annealing neural network for convex programming. Neural Networks 7(4), 629–641 (1994)
12. Hu, X., Zhang, B.: A new recurrent neural network for solving convex quadratic programming problems with an application to the k-winners-take-all problem. IEEE Trans. Neural Netw. 20(4), 654–664 (2009)
13. Hu, X., Zhang, B.: An alternative recurrent neural network for solving variational inequalities and related optimization problems. IEEE Trans. Syst., Man, Cybern. B 39(6), 1640–1645 (2009)
14. Cheng, L., Hou, Z.G., Tan, M.: A delayed projection neural network for solving linear variational inequalities. IEEE Trans. Neural Netw. 20(6), 915–925 (2009)
15. Forti, M., Nistri, P., Quincampoix, M.: Generalized neural network for nonsmooth nonlinear programming problems. IEEE Trans. Circuits Syst. I 51(9), 1741–1754 (2004)
16. Liu, Q., Wang, J.: A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming. IEEE Trans. Neural Netw. 19(4), 558–570 (2008)