Volume : 2 | Issue : 5 | May 2013 • ISSN No 2277 - 8160

Research Paper | Computer Science

# Optimizing Single-Layer Raster Cellular Neural Network Simulator Using Simulated Annealing Technique with Rk84(5)

**O.H. Abdelwahed** — Department of Computer Science, Faculty of Computers and Information Suez Canal University, EGYPT

**M. El-Sayed Wahed** — Department of Computer Science, Faculty of Computers and Information Suez Canal University, EGYPT

**ABSTRACT**

*An efficient numerical integration algorithm for single layer Raster Cellular Neural Networks (CNN) simulator is presented in this paper. Explicit Runge{Kutta (RK) methods provide an attractive means for the solution of initial value problems of first-order differential equations in the form of pairs of orders p (p -1). Most existing RK formulas (single methods as well as pairs) use the minimal number of stages required for achieving a prescribed order. In this article we shall study, in terms of efficiency and reliability, RK pairs of orders p (q).The simulator is capable of performing CNN simulations for any size of input image, thus a powerful tool for researchers investigating potential applications of CNN. This paper reports an efficient algorithm exploiting the latency properties of Cellular Neural Networks along with numerical integration techniques; simulation results and comparisons are also presented.*

**KEYWORDS: single-layer Cellular neural networks, numerical integration algorithms, RK84(5)**

## 1 Introduction

CNN is a hybrid of Cellular Automata and Neural Networks (hence the name Cellular Neural Networks), and it shares the best features of both worlds. Like Neural Networks, its continuous time feature allows real-time signal processing, and like Cellular Automata, its local interconnection feature makes VLSI realization feasible. Its grid-like structure is suitable for the solution of a high order system of first order non-linear differential equations on-line and in real-time. CNN is an analog nonlinear dynamic processor array; characterized by the following features [3]:

1) Each analog processor is capable of processing continuous signals, in either continuous-time or discrete-time modes.
2) The processors are placed on a 3D geometric cellular grid (several 2D layers) and are basically identical.
3) Interaction among processors is local and mainly translation invariant.
4) The mode of operation may be transient, equilibrium, periodic, chaotic, or combined with logic (without A/D conversion).

The basic circuit unit of CNN is called a [2]. It contains linear and nonlinear circuit elements. Any cell, C(i,j), is connected only to its neighbor cells, i.e. adjacent cells interact directly with each other. This intuitive concept is called neighborhood and is denoted as N(i,j). Cells not in the immediate neighborhood have indirect effect because of the propagation effects of the dynamics of the network. Each cell has a state x input U, and output y. The state of each cell is bounded for all time t > U and, after the transient has settled down, a cellular neural network always approaches one of its stable equilibrium points. This last fact is relevant because it implies that the circuit will not oscillate. The dynamics of a CNN has both output feedback (A) and input control (B) mechanisms. The first order nonlinear differential equation defining the dynamics of a cellular neural network cell can be written as follows:

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N(i,j)} A(i,j;k,l)y_{kl}(t) + \sum_{C(k,l)\in N(i,j)} B(i,j;k,l)u_{kl}$$

$$y_{ij}(t) = \frac{1}{2}\left(|x_{ij}(t)+1| - |x_{ij}(t)-1|\right) \qquad (1)$$

where $x_{ij}$ is the state of cell C(i,j), $x_{ij}(0)$ is the initial condition of the cell, C is a linear capacitor, R is a linear resistor, I is an independent current source, A(I,J;k, 1)$y_{kl}$ and B{i,j;k,l)$u_{kl}$ are voltage controlled current Sources for all cells C(k,l) in the neighborhood N(ij) of cell C(ij), and yij represents the output equation.

Notice from the summation operators that each cell is affected by its neighbor cells. A(.) acts on the output of neighboring cells and is referred to as the feedback operator. B(.) in turn affects the input control and is referred to as the control operator. Spe- cific entry values of matrices A(.) and B(.) are application dependent, are space invariant and are called cloning templates. A current bias Z and the cloning templates determine the transient behavior of the cellular nonlinear network. The equivalent block diagram of a continuous-time cell implementation is shown in Fig.lb.

CNNs have as input a set of analog values and its programmability is done via cloning templates.Thus, programmability is one of the most attractive properties of CNNs, but how to choose the optimal network and how to program it to perform a given task are still topics under investigation. This is the reason why there is a need for behavioral CNN simulator capable of helping investigators design and manipulate cloning templates ("programming"). Existent tools are not meant to deal with a significant number of pixels typical in common image processing applications [5]. The simulator presented here not only satisfies this need, but it also can be used for testing CNN hardware implementations. M. El-Sayed Wahed and O.H. Abdel wahed[1] introduced an efficient numerical integration algorithm for Single-Layer Raster Cellular Neural Networks Simulator. In this paper, we consider the same problem since we optimize their solution by using the optimization technique, simulated annealing.

## 2 Behavioral Simulation

Recall that equation (1) is space invariant, which means that A(i,j;k,l) = A(i-k,j-1) and B(i,j;k,l) = B(i,k;,j,l) for all i,j,kl.

Therefore, the solution of the system of difference equations can be seen as a convolution process between the image and the CNN processors. The basic approach is to imagine a square subimage area centered at (x,y), with the subimage being the same size of the templates involved in the simulation. The center of this subimage is then moved from pixel to pixel starting, say, at the top left comer and applying the A and B templates at each location (x,y) to solve the differential equation. This procedure is repeated for each time step, for all the pixels. An instance of this image scanning-processing is referred to as an "iteration". The processing stops when it is found that the states of all CNN processors have converged to steady-state values[2] and the outputs of its neighbor cells are saturated, e.g. they have a +1 value.

This whole simulating approach is referred to as raster simulation. A simplified algorithm is presented below for this approach. The part where the integration is involved (i.e. calculation of the next state) is explained in the Numerical Integration Methods section.

In the following two subsections we will discuss simulated annealing algorithm and the mathematical modeling used in simulated annealing and our proposed simulator.

### 2.1 The Simulated Annealing Algorithm

In the early 1980s Kirkpatrick et al. (1983) and independently Cemy (1985)[7] introduced the concepts of annealing in combinatorial optimization. Originally these concepts were heavily inspired by an analogy between the physical anneahng process of solids and the problem of solving large combinatorial optimization problems. Since this analogy is quite appealing we use it here as a background for introducing simulated annealing.In condensed matter physics, anneahng is known as a thermal process for obtaining low energy states of a sohd in a heat bath. The process consists of the following two steps:

- increase the temperature of the heat bath to a maximum value at which the solid melts;
- decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid.

In the liquid phase, all particles arrange themselves randomly, whereas in the ground state of the solid, the particles are arranged in a highly structured lattice, for which the corresponding energy is minimal. The ground state of the solid is obtained only if the maximum value of the temperature is sufficiently high and the cooling is performed sufficiently slowly. Otherwise, the solid will be frozen into a meta-stable state rather than into the true ground state.

Metropolis et al.[7] introduced a simple algorithm for simulating the evolution of a solid in a heat bath to thermal equilibrium. Their algorithm is based on Monte Carlo techniques (Binder, 1978) and generates a sequence of states of the solid in the following way.

Given a current state i of the solid with energy $E_i$, then a subsequent state j is generated by applying a perturbation mechanism which transforms the current state into a next state by a small distortion, for instance by displacement of a particle. The energy of the next state is $E_j$. If the energy difference, $E_j$— $E_i$, is less than or equal to zero, the state j is accepted as the current state. If the energy difference is greater than zero, then the state j is accepted with a probability given by $\exp\left(\frac{E_i - E_j}{k_B T}\right)$ where T denotes the temperature of the heat bath and kB is a physical constant called the Boltzmann constant. The acceptance rule described above is known as the Metropolis criterion and the algorithm that goes with it is known as the Metropolis algorithm. It is known that, if the lowering of the temperature is done sufficiently slowly, the solid can reach thermal equilibrium at each temperature. In the Metropohs algorithm this is achieved by generating a large number of transitions at a given value of the temperature. Thermal equilibrium is characterized by the Boltzmann distribution, which gives the probability of the solid of being in a state / with energy Ei at temperature T, and which is given by

$$P_T\{X = i\} = \frac{\exp(-E_i / k_B T)}{\sum_j \exp(-E_i / k_B T)}$$

where X is a random variable denoting the current state of the solid and the summation extends over all possible states. As we indicate below, the Boltzmann distribution plays an essential role in the analysis of the convergence of simulated annealing. Returning to simulated annealing, the Metropolis algorithm can be used to generate a sequence of solutions of a combinatorial optimization problem by assuming the following equivalences between a physical many-particle system and a combinatorial optimization problem:

- solutions in the combinatorial optimization problem are equivalent to states of the physical system;
- the cost of a solution is equivalent to the energy of a state.

Furthermore, we introduce a control parameter which plays the role of the temperature. Simulated annealing can thus be viewed as an iteration of Metropolis algorithms, executed at decreasing values of the control parameter.

We can now let go of the physical analogy and formulate simulated annealing in terms of a local search algorithm. To simplify the presentation, we assume, in the remainder of this chapter, that we are dealing with a minimization problem. The discussion easily translates to maximization problems. For an instance (S, f) of a combinatorial optimization problem and a neighborhood function .The meaning of the four functions in the below procedure in fig. 2. is obvious: INITIALIZE computes a start solution and initial values of the parameters c and L; GENERATE selects a solution from the neighborhood of the current solution; CALCULATE.LENGTH and CALCULATE_CONTROL compute new values for the parameters L and c, respectively.

As already mentioned, a typical feature of simulated annealing is that, besides accepting improvements in cost, it also accepts deteriorations to a limited extent. Initially, at large values of c, large deteriorations will be accepted; as c decreases, only smaller deteriorations will be accepted and, finally, as the value of c approaches 0, no deteriorations will be accepted at all.

```
procedure SIMULATED ANNEALING;
begin
INITIALIZE (istart, Co, Lo);
k:=0;
repeat
for i := 1 to L_k do
begin
GENERATE (j from S_i);
i f f(j) ≤ f(i) then i := j
else
if exp((f(i) − f(j))/c_k) > random[0, 1) then i := j
end;
k:= k + 1;
CALCULATE_LENGTH (Lk);
CALCULATE_CONTROL(ck);
until stopcriterion
end;
```

### 2.2The Mathematical Model and the proposed simulator

Simulated annealing can be mathematically modeled by means of Markov chains. In this model, we view simulated annealing as a process in which a sequence of Markov chains is generated, one for each value of the control parameter. Each chain consists of a sequence of trials, where the outcomes of the trials correspond to solutions of the problem instance.

Let {S, f) be a problem instance, N a neighborhood function, and X(k) a stochastic variable denoting the outcome of the kth trial. Then the transition probability at the ^th trial for each pair i, j € S of outcomes is defined as

$$\begin{cases} \{G_{ij}(c_k A_j(c_k)) f & i \neq j \\ 1 - \sum_{t \in S, l \neq i} \{G_j(c_k A_j(c_k)) f & i = j \end{cases}$$

Pij{k) = P{X(k) = j|X(k-1) = i} =

where $G_{ij}$ (ck) denotes the generation probability, i.e. the probability of generating a solution j when being at solution i, and $G_{ij}$ (ck) denotes the acceptance probability, i.e. the probability of accepting solution j , once it is generated from solution i. The most frequently used choice for these probabilities is the

**following :**

$G_{ij}$ (ck) =

$$\begin{cases} |N(i)|^{-1} & f \quad j \in S_i \\ 0 & f \quad j \notin S_i \end{cases}$$

And

$A_j(c_k)$ =

$$\begin{cases} 1 & f \quad f(j) \leq f(i) \\ \exp((f(i) - f(j)) / c) & f \quad f(j) > f(i) \end{cases}$$

For fixed values of c, the probabilities do not depend on k, in which case the resulting Markov chain is time-independent or homogeneous. Using the theory of Markov chains it is fairly straightforward to show that, under the condition that the neighborhoods are strongly connected—in which case the Markov chain is irreducible and periodic—there exist a unique stationary distribution of the outcomes. This distribution is the probability distribution of the solutions after an infinite number of trials.

Algorithm: (Single-Layer or Raster CNN simulation with Simulated Annealing)
Obtain the input image, initial conditions and templates from user;
/* M,N = # of rows/columns of the image */
/* APPLY SIMULATED ANNEALING */
begin
INITIALIZE (istart, Co, Lo);
k:=0;
repeat
for / := 1 to Lk do
begin
GENERATE (j from Si);
i f f(j) $\leq$ f(i) then i := j
else

$$\left(\frac{f(i)-f(j)}{c_k}\right)$$

if exp > random[0, 1) then i := j
end;
k:= k + 1;
CALCULATE_LENGTH (Lk);
CALCULATE_CONTROL(ck);
until stopcriterion
end;
/* Use the optimized parameters from the simulated annealing */
while (converged-cells < total # of cells) (
for (i=l; i<=M; i++)
for (j=l; j<=N; j++) (
if (convergence-flag[i] [j])
/* calculation of the next state*/
continue; /* current cell already converged */

$$x_j(t_{n+1}) = x_j(t_n) + \int_{\hat{o}_n}^{\hat{o}_{n+1}} f(x(n\hat{o}))d\hat{o}$$

/* convergence criteria */

$$if\left(\frac{dx_{ij}}{dt}=0 \text{ and } y_{ij}=\pm 1 \forall C(k,l)=N_r(i,j)\right)$$

{
convergence-flag[i]fi] = 1;
converged-cells++ ;
}
} I* end for *I
/* update the state values of the whole image*/
for (i=l; i<=M; i++)
for (j=l; j<=N; j++) (
if (convergence-flag[i]h l) continue;
$X_{ij}(t_{n+1}) = X_{ij}(t_n)$;
}
#-ofjteration++;
) I* end while *I
The raster approach implies that each pixel is mapped onto a CNN processor. That is, we have an image processing function in the spatial domain that can be expressed as:
g(x,y) = T(f(x,y)) (2)
where f(.) is the input image, g(.) the processed image, and T is an operator on f(.) defined over the neighborhood of (x,y).

## 3 Numerical Integration Methods

M. El-Sayed Wahed and O.H. Abdel wahed [1] introduced An efficient numerical integration algorithm for Single-Layer Raster Cellular Neural Networks Simulator. We consider the same problem but by using three of the single-step numerical integration algorithms used in the CNN behavioral simulator described here. They are RK4(2), RK4(3), and RK84(5) algorithms.

### 3.1.1 Stepsize selection algorithm.

There are currently two widely used methods that have appeared in the literature for changing the stepsize of p (q)-order RK codes. The first is to apply the formula (see [9])

$$h_{n+1} = f\left(\frac{TOL}{EST_n}\right)^{\frac{1}{p}} h_n \quad (3)$$

Where f1 is a safety factor and the new sought-after stepsize $h_{n+1} = x_{n+1} - x_n$ is predicted in terms of an estimate of the local error $EST_n$ which is based on the approximation

$$EST_n \approx y_n - y_n \quad (4)$$

Assuming $y_n$, $\hat{y}_n$ to be the pth-, qth-order approximate solutions, respectively, at the previous grid point xn and TOL the requested tolerance. If

$$EST_n \leq TOL,$$

Then the computed solution yn+1 is accepted and the integration is carried out, otherwise(5) is reevaluated by substituting

$$EST_n \rightarrow EST_{n+}$$

This methodology is termed the error per step (EPS) mode (see Shampine [10]).

An alternative is to consider the same algorithm (5), but to use, instead of (6), the approximation

$$EST_n \cong \frac{y_n - y_n}{h_n}, \quad (5)$$

This is called error per unit step (EPUS) [10].

### 3.1.2 RK4(2) and RK4(3) at n = 4

**According to [8], The equations of RK4(2) and RK4(3) can be deduced from the following table:Table 1**

| | | | | | |
|---|---|---|---|---|---|
| 0 | | | | | |
| $\frac{5}{14}$ | $\frac{5}{14}$ | | | | |
| $\frac{13}{22}$ | $-\frac{52}{605}$ | $\frac{819}{1210}$ | | | |
| 1 | $\frac{2576}{4745}$ | $-\frac{252}{365}$ | $\frac{1089}{949}$ | | |
| 1 | $\frac{19}{130}$ | $\frac{343}{1215}$ | $\frac{1331}{3159}$ | $\frac{73}{486}$ | |
| 4th order | $\frac{19}{130}$ | $\frac{343}{1215}$ | $\frac{1331}{3159}$ | $\frac{5}{14}$ | |
| 2nd order | $\frac{4}{55}$ | $\frac{203}{990}$ | $\frac{13}{18}$ | | |
| 3rd order | $\frac{11}{130}$ | $\frac{637}{1215}$ | $\frac{605}{3159}$ | $-\frac{73}{243}$ | $\frac{1}{2}$ |

**Table 1: Coefficients of the pairs NEW4(2) and NEW4(3)**

Therefore, the final integration is a weighted sum of the five calculated-

derivates is given: xij((n+1)τ) =xij(nτ) + $\frac{19}{130}$ k1ij + $\frac{203}{990}$k2ij + $\frac{1331}{3159}$ k3ij + $\frac{73}{486}$k4ij (6)

The difference between Rk4(2) and RK4(3) is the local truncation error in the case of RK4(2) is given by using the RK(2)i.e.

$y_{ij}((n+1)\tau) = y_{ij}(n\tau) + \frac{4}{5} k^1_{ij} + \frac{343}{1215} k^2_{ij} + \frac{637}{1215} k^3_{ij}$ (7)

But local truncation error in the case of RK4(3) is given by using the RK(3)i.e.

$y_{ij}((n+1)\tau) = y_{ij}(n\tau) + \frac{1}{130} k^1_{ij} + \frac{637}{1215} k^2_{ij} + \frac{605}{3159} k^3_{ij} - \frac{73}{1215} k^4_{ij}$ (8)

### 3.1.3 RK84(5) at n= 8

According to [8] , The equations of RK 8(5) are:

$k1ij = \tau f'xij(n\tau)$

$k2ij = \tau f'(xij(n\tau)) - \frac{1}{3}\, k1ij$

$k3ij = \tau f'(xij(n\tau)) + \frac{78991}{290322}\, k1ij - \frac{46225}{290322}\, k2ij$

$k4ij = \tau f'(xij(n\tau)) + \frac{3}{1016}\, k1ij + \frac{129}{1016}\, k3ij .$

$k5ij = \tau f'(xij(n\tau)) + \frac{1697713059}{4222509269}\, k1ij - \frac{15238032203}{10156496298}\, k3ij + \frac{11056598884}{7292015089}\, k4ij$

$k6ij = \tau f'(xij(n\tau)) + \frac{55431}{107844}\, k1ij + \frac{2048383}{8149188}\, k4ij + \frac{1953125}{9902211}\, k5ij$

$k7ij = \tau f'(xij(n\tau)) - \frac{9684214799}{14765520605}\, k1ij + \frac{956894283}{7559277968}\, k4ij - \frac{465115410}{11816446109}\, k5ij + \frac{91302285}{4596652571}\, k6ij$

$k8ij = \tau f'(xij(n\tau)) - \frac{134489695}{1465284848}\, k1ij - \frac{956689878700}{6901605883}\, k4ij - \frac{343998932833}{12958171610}\, k5ij + \frac{254650197888}{105790165299}\, k6ij + \frac{769862021250}{5122674515}\, k7ij$

$k9ij = \tau f'(xij(n\tau)) + \frac{145536625}{3474014636}\, k1ij - \frac{1303358981}{1702211673}\, k4ij - \frac{55898639}{2339992721}\, k5ij + \frac{921475172}{7161215321}\, k6ij + \frac{1102593162270}{1022467820}\, k7ij - \frac{80727265}{11312405923}\, k8ij$

$k10ij = \tau f'(xij(n\tau)) - \frac{13686537520080}{3365041800}\, k1ij - \frac{19151417051}{2883993186}\, k4ij - \frac{22521917029}{12057970022}\, k5ij - \frac{953123275033}{22272368203}\, k6ij + \frac{32094737450}{83875934630}\, k7ij - \frac{16775244890}{6391208017}\, k8ij$



$k11ij = \tau f'(xij(n\tau)) - \frac{11959297910}{15149569322}\, k1ij - \frac{35554033801}{2078515654}\, k4ij + \frac{8903076353}{16738414228}\, k5ij - \frac{807813783170}{13468382457}\, k6ij + \frac{2810110890320}{148656749130}\, k7ij + \frac{19747907810}{11858655590}\, k8ij + \frac{2054461365900}{351296882400}\, k9ij + \frac{75021976250}{30319520681}\, k10ij$

Therefore, the final integration is a weighted sum of the eleven calculated derivates is given:

$xij((n+1)\tau) = xij(n\tau) + \frac{1}{10}\, k1ij + \frac{7}{66}\, k6ij + \frac{9}{180}\, k7ij + \frac{1}{5}\, k8ij + \frac{1}{5}\, k9ij + \frac{3}{180}\, k10ij + \frac{1}{10}\, k11ij$

The difference between RK8(5) and RK8(4) is that the error in the case of RK8(5) is computed by using the following equation:

$yij((n+1)\tau) = yij(n\tau) + \frac{1}{10}\, k1ij - \frac{9}{44}\, k6ij + \frac{9}{180}\, k7ij + \frac{1}{5}\, k8ij + k9ij + \frac{3}{180}\, k10ij + \frac{1}{10}\, k11ij$

While in the case f RK8(4), the error is computed by the following equation:

$yij((n+1)\tau) = yij(n\tau) + \frac{2350230046}{4905448450 1}\, k1ij + \frac{3649218174}{134615774990}\, k4ij + \frac{545839447}{8942617608 7}\, k5ij + \frac{4566413657}{2990851576 1}\, k6ij + \frac{1}{5}\, k8ij + \frac{1}{5}\, k9ij \frac{3}{180}\, k10ij + \frac{1}{10}\, k11ij$

Where f(l.) is computed according to (1). There are many single step methods available to us for this purpose. But, one option worth considering is the combination of two methods in solving for the solution.

So we use RK84(5) and RK8(5) to make a very efficient computer solving the problem the way it evaluates the integral presented

## 4 Simulation Results and Comparisons

The simulation time used for comparisons is the actual CPU time used. The input image format for this simulator is a JPEG format.



**Fig.2. Image processing (a) After Averaging Template (b) After Averaging and Edge Detection**

Fig. 2 shows results of the raster simulator obtained from a complex image of 65,536(256x256) pixels. For this example an Averaging template followed by an Edge Detection template were applied to the original image to yield the images displayed in Figs. 2a and 2b, respectively.

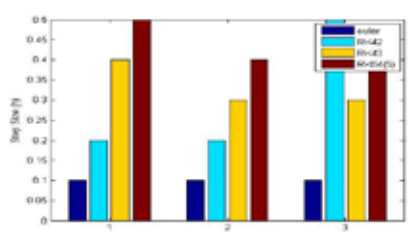
Also in figure 3, it has been shown the quality

**Fig.3. Quality measures of fig.2a and fig.2b**

| method | Mean Square Error | Peak Signal to Noise Ratio | MNormalized Cross-Correlation | Average Difference | Structural Content | Maximum Difference | Normalized Absolute Error |
|---|---|---|---|---|---|---|---|
| RK4(2) | 1.6770e +003 | 12.3931 | 0.9355 | 8.9990 | 1.5253 | 241 | 0.0750 |
| RK4(3) | 1.6995e +003 | 11.3930 | 0.9100 | 6.2440 | 1.3111 | 233 | 0.0480 |
| RK84(5) | 1.6800e +003 | 10.3937 | 0.9600 | 6.0042 | 1.7000 | 234 | 0.0360 |

measures of the two pictures in 2a and 2b by using the numerical techniques:

RK84(5) using simulated annealing. We notice that these results are better than those in the literature. Since speed is one of the main concerns in the simulation, finding the maximum step size that still yields convergence for a template can be helpful in speeding up the system. The speed-up can be achieved by selecting an appropriate Δt for that particular template. Even though the maximum step size may slightly vary from one image to another, the values in Fig.4 still serve as good references. These results were obtained by trial and error over more than 100 simulations On Lena image with small size 43x64(2752 pixels).The importance of selecting an appropriate Δt can be easily visualized in Fig. 4. If the step size chosen is too small, it might take many iterations, hence longer time, to achieve convergence. On the other hand, if the step size taken is too large, it might not converge at all or it would converge to erroneous steady state values. The results of Fig. 5 were obtained by simulating Lena image of size 43x64(2752 pixels).



Edge Detection Averaging Connected Component

**Fig.4. Maximum step size that still yield convergence for 4 different templates**

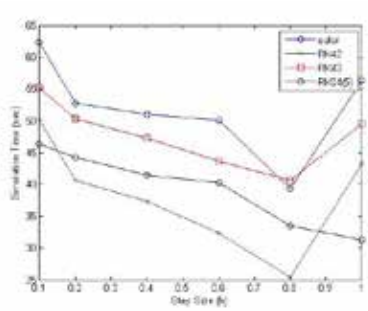


**Fig.5. Simulation time comparisons for 4 different numerical techniques**

for four different templatesusing an Edge detection template. We notice that the CPU time for our method is better than those in the literature.

**5 Conclusion**

As researchers are coming up with more and more CNN applications, an efficient and powerful simulator is needed. So we use simulated annealing in optimizing CNN using the numerical integrations, especially using RK84(5) for more efficiency. The simulator hereby presented meets the need in six ways: 1) Depending on the accuracy required for the simulation, the user can choose from three numerical methods to perform the numerical integration, 2) The input image format is JPEG, which is commonly available, 3) The input image can be of any size, allowing simulation of images available in common practices, 4) CPU time of our methods is better than those in the literature, 5 ), the quality measures of the pictures and the edge detection for our method is better than those in the literature

## REFERENCES

[1] M. El-Sayed Wahed and O.H. Abdel wahed (2012). An efficient numerical integration algorithm for Single-Layer Raster Cellular Neural Networks Simulator. the international Journal of the physical sciences(IJPS), December 16th 2012. | [2] L. 0. Chua and L. Yang(1988). "Cellular Neural Networks: Theory & Applications," IEEE Trans. Circuits and Systems, Vol. CAS-35, pp. 1257-1290. | [3] L.O. Chua and T. Roska(1992). "The CNN Universal Machine Part 1: The Architecture", in Int. Workshop on Cellular Neural Networks and their Applications (CNNA), pp. 1-10. | [4] J. A. Nossek, G. Seiler, T. Roska and L. 0. Chua (1992.). "Cellular Neural Networks: Theory and Circuit Design," International Journal of Circuit Theory and Applications, Vol. 20, pp. 533-553. | [5] J. Varrientos and E. Sanchez-Sinencio(1992), "CELLSIM: A cellular neural network simulator for the personal computer," in Proc. 35th Midwest Symp. Circuits Systs, pp. 1384-1387. | [6] W. H. Press, B. P. Flannery, S.A. Teukolsky, and W.T.g Vetterling(1986). "Numerical Recipes. The Art of Scientific Computing", Cambridge University Press, New York. | [7] P.J.M. van Laarhoven and E.H.L. Aarts, (1987) Simulated Annealing: Theory and Application. ISBN 90-277-2513-6 | [8] Ch. Tsitouras and S. N. Papakostas(1991) "Cheap Error methods for Runge-Kutta methods ", SIAM J. SCI. COMPUT, Society for Industrial and Applied Mathematics, Vol. 20, No. 6, pp. 2067-2088. | [9] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick(1972), Comparing numerical methods for ordinary differential equations, SIAM J. Numer. Anal., 9 , pp. 603{637. | [10] L. F. Shampine(1986), Some practical Runge-Kutta formulas, Math. Comp., 46 , pp. 135{150. |