1 Introduction
Segmentation is a topic well researched in computational vision. Its main goal is dividing a picture to a set of segments; each holds a set of pixels from that picture, each of which are related to each other. Deciding correctly which pixel is related to which segment is difficult, and also which characteristics are to base the decision on is not clear.
Several algorithms were proposed for this task. Among them are relaxation and segmentation using clustering.
Here I proposed another method: Segmentation using Simulated Annealing methods, specifically the Metropolis Algorithm.

2 The Metropolis Algorithm
Given an energy function $f: \mathbb{R}^n \rightarrow \mathbb{R}$, this algorithm gives its global minimum, regardless if the function is derivable or continuous. It is based on the thermodynamics basic assumption, stating that the probability of a system of many particles to switch from a state A to a state B is proportional to the expression:

$$e^{-\frac{(E_x-E_A)}{\tau}}$$

where $\tau$ is the temperature. You can see that switching from a state with high energy to one with lower energy will always occur (Probability larger than 1), and switching to a state with higher energy is also possible, where the probability decreases as the temperature drops.
Also it is using the observation that a hot system that cools slowly will set itself in a state of low energy (Imagine a liquid that is allowed to freeze slowly until it is perfectly crystallized).
The algorithm itself randomly chooses a point in an n-dimensional space, than iterating until it is finished. On each iteration it chooses another point, calculates its energy and using the above expression, deciding if it should move to the new state, and thus allowing sometimes moving to a state with higher energy in order to escape a local minimum. After several iterations it lowers the temperature and thus allowing cooling and finally setting the right point.
Deciding the parameters of the algorithm isn't obvious. The cooling rate and the initial temperature should be set by trial and mistake. I set the cooling rate to be 10 percent decrease each 20-30 iterations. Also I chose the initial temperature by randomly take 5 points, and choose the energy of the point with highest energy as the initial temperature.
Stopping condition is also not obvious. Some choose stopping condition to be an arbitrary number of iterations. I chose it to be less than 5 percent of energy change for five iterations.
This algorithm showed excellent results in travelling salesman problem and other optimization problems.
Its pseudo code is shown below:
while not finished
    if energy isn't changed for the 5th time
        finish
    else
        choose a random point $P_{\text{new}}$ and compare to current point $P_{\text{current}}$
        the distance to the new point is proportional to $T/T_{\text{max}}$
        if $E(P_{\text{new}}) < E(P_{\text{current}})$
            switch to the new point
            if $E(P_{\text{new}}) < E(P_{\text{best}})$
                choose $P_{\text{new}}$ to be $P_{\text{best}}$
        else
            choose a random number $m$
            if $m < \exp(-E(P_{\text{new}}) - E(P_{\text{current}})/T)$
                switch to the new point

if this is the 20'th iteration
    decrease $T$ by 10 percent

3 Implementation
A point here is the center of a segment. It is defined in an 11-dimensional space, consisted of $x$ and $y$ value, $R$, $G$ and $B$ values and $x$ and $y$ values of the gradients of $R$, $G$ and $B$ layers.
Indexing of the pixels is done by getting each pixel's attributes and indexing it to the nearest center. Its energy is its distance from that center. The energy function who's minimum is to find is the sum of distances of all pixels.
I compared the algorithm to another known algorithm: Segmentation using clustering. The comparison was both subjective with the quality of the segmentation itself, and objective: Comparing the number of indexing processes.

4 Results
Below are the results of applying the algorithm on several pictures.

<table>
<thead>
<tr>
<th>Shapes, differ only by color</th>
<th><img src="image" alt="Shapes" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation using Metropolis, 353 iterations</td>
<td><img src="image" alt="Segmentation" /></td>
</tr>
</tbody>
</table>
Segmentation using Clustering, 4 iterations

Picture differ by only gradient

Segmentation using Metropolis with 4 segments, 293 iterations

Segmentation using Clustering with 4 segments, 4 iterations
Segmentation using Metropolis with 5 segments, 236 iterations

Segmentation using Clustering with 5 segments, 3 iterations

Segmentation using Metropolis with 6 segments, 271 iterations

Segmentation using Clustering with 6 segments, 4 iterations
5 Conclusion

According to literature the algorithm is efficient relatively to other optimization algorithms. It also has the ability to 'escape' local minima in order to find the global maximum. In literature there is also a proof for convergence of the algorithm to the real minimum, but it is not said how and how fast.

Here it is shown that the algorithm has poor performance relatively to the reference one.

There are some reasons for that:

- The finding the exact centers of the segments is less important than converging to it. In other word there are a lot of local minima which are equally satisfying.
- The tests were made on simple pictures were the Metropolis algorithm didn't have a chance to show its strength.

Also there are some questions about segmentation that should be asked and weren't completely dealt here, and I also didn't find reference to them in other places:

- **Deciding the number of segments.** In most pictures it is not obvious how many segments are in the picture. We can treat this as another degree of freedom, but this degree is not equally in weight to other degrees and also,
because the energy will always be lower as the number of segments is higher, regardless the correctness of the choice.

- **Weight of each degree of freedom.** Each variable is measured in a completely different set of units and range, so the naive metric: \( \sqrt{\sum x_i^2} \) is wrong and instead we should use something like \( \sqrt{\sum f(x_i)^2} \). Finding these functions is not clear and one may try the easiest possibility: \( \sqrt{\sum \alpha_i x_i^2} \) where the weights are arbitrary. Using again the metropolis algorithm with these weights as more dimensions to the space caused the tests to run for more than a day. Also in different pictures there are different weights.

- **Parameters to search.** As mentioned above, the parameters of the space were x and y values, RGB values and gradient values. Other parameters such as general intensity, gradient in polar coordinates, etc.

- **Calibration.** Finally I would like to emphasize the main disadvantage of this algorithm. It is mainly used to calibrate systems, but in order to apply it you have to calibrate it first. You get a situation where you have to calibrate the calibration, which is the main obstacle of this algorithm.