Optimising Monte Carlo Search Strategies for Automated Pattern Detection

by Jeffrey S. Rosenthal*

(June 2008; comments welcome.)

1. Introduction.

Automated pattern detection is a well-studied area of computer vision (see e.g. [3], [1], [2], and the references therein), with obvious importance for computer vision and artificial intelligence. Among other things, it presents challenging problems in statistical computation, requiring Monte Carlo and other sophisticated search strategies to efficiently explore large parameter spaces.

In this short paper, we consider the relatively simple problem of accurately detecting a "face" (two eyes and a nose) from a sea of pixels. We describe an interactive pattern-detection Java applet [7]. We present a simple score function for facilitating pattern detection, and various Monte Carlo algorithms for attempting to maximise it. We report simulation experiments to investigate various algorithm choices, and determine which choices lead to most efficient score optimisation. We also develop a simple theoretical framework for approximately optimising one of the parameters in the algorithm.

2. Experimental Framework.

We use the Java applet [7] to conduct experiments on the various search strategies. We use a test image as in Figure 1, consisting of a 125×95 grid of pixels (some on and some off), which we regard as a "blurry" image including two eyes and a nose plus lots of noise.

The challenge for the computer algorithm is to find the face within the image, given only the image pixel values. A successful search (Figure 2) finds the face fairly accurately, placing the eyes and nose in nearly their correct location. On the other hand, an unsuccessful search (Figure 3) miscalculates the face location, in this case placing the nose where the right eye

*Department of Statistics, University of Toronto, Toronto, Ontario, Canada M5S 3G3. Email: jeff@math.toronto.edu. Web: probability.ca/jeff. Supported in part by NSERC of Canada.

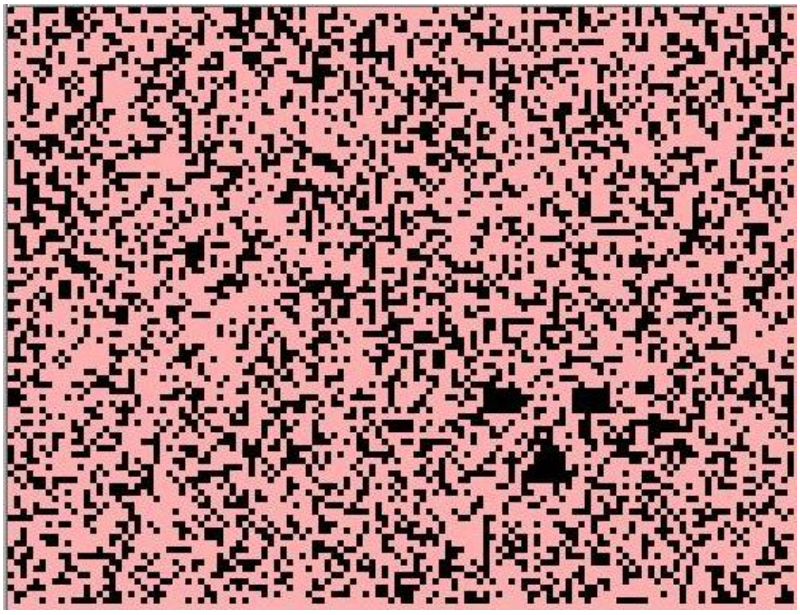

Figure 1. A test pattern for a face location search.

should be. Between these two extremes, a partially-successful search might place the eyes and nose in approximately the correct location, but somewhat off-center or wrong sized or otherwise inaccurate.

So, now the question becomes, which computer search strategies will lead to more successful searches and fewer unsuccessful ones?

3. A Pattern-Recognition Model.

To implement a search algorithm, we require a model for what constitutes a successful or unsuccessful search. We do this by means of a score function S, depending on various parameters, which indicates the extent to which parameters do or do not accurately describe the location of an object.

We begin by describing a general framework, applicable to any objects.

3.1. General Framework.

We focus on a very simple model of pattern recognition (following the advice of [1], p. xiii, to keep object detection models "as simple as possible").

We assume that we seek some object (e.g., a face), and that the object is approximated by some specified object region depending on various parameters (e.g. eye width, nose height,

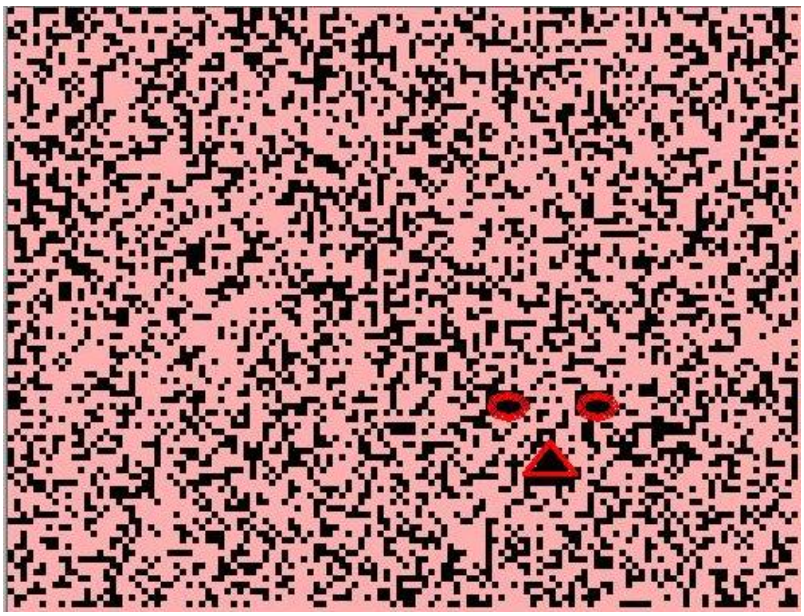

Figure 2. A successful search result (red).

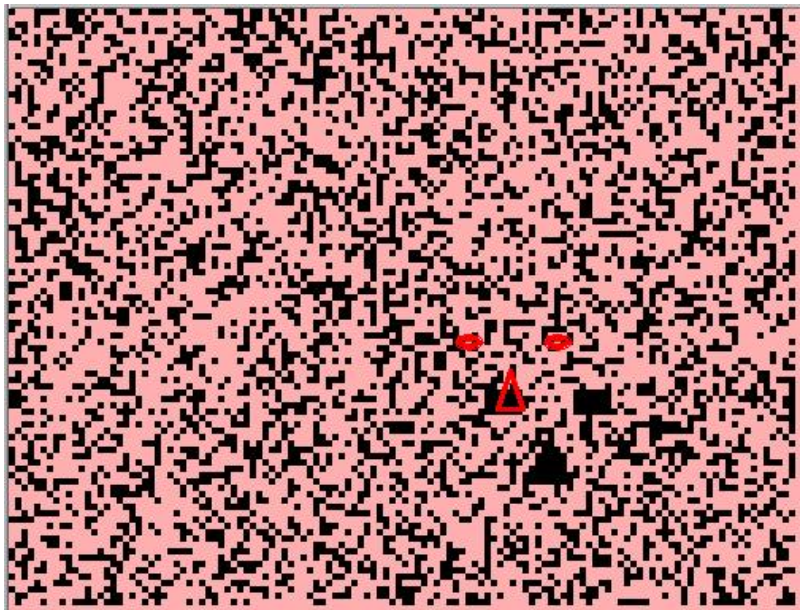

Figure 3. An unsuccessful search result (red).

etc.).

We measure the evidence for an object with particular parameters in a particular location by means of an extremely simple score function, namely the number of activated pixels minus twice the number of unactivated pixels within the specified object region. (This model thus gives highest scores to large object regions which are mostly activated, though the "twice" factor is included to discourage gratuitously large potential objects. In practice [7], this model seems to work as well or better than the more complicated statistical models we have considered involving likelihood ratios, etc.) Any pixel outside of the available image range is treated as being unactivated.

This general framework could apply to detection of any objects. To specify a model for particular objects (e.g. faces), it is necessary to list the parameters being allowed, and the object region corresponding to those parameters. We do that next.

3.2. Faces Model.

We model a face (two eyes and a nose) in terms of seven parameters x, y, w, s, e, h, b, as follows. The location (origin) is at (x, y). The parameter w represents the width of the face, so the centers of the eyes are at $(x \pm w, y)$. The parameter e represents the size of the eyes, so the eyes each have width $2e$ and height e. The parameter s represents the separation of the nose from the eyes, so the top of the nose is at $(x, y - s)$. The parameters h and b represent the height and breadth of the nose, so the nose is a triangle with top virtex at $(x, y - s)$, and bottom vertices at $(x \pm b, y - s - h)$. These seven parameters then define a corresponding object region, as shown in Figure 4.

The score function in terms of these seven parameters, $S(x, y, w, s, e, h, b)$, is then given, as above, by the number of activated pixels minus twice the number of unactivated pixels within the corresponding object region.

To avoid illogical special cases, we make the following restrictions on the parameter values (by returning a score of $-\infty$ if they are violated): $1 \leq e < w \leq 20$, $0 \leq s \leq w$, $0 \leq h \leq 20$, $0 \leq b \leq w$, and (so the nose extends below the eyes) $s + h > e/2$. Any parameter configuration violating these restrictions will be referred to as *out-of-bounds*. We further assume that all parameters are integers.

We do not explicitly restrict the center point (x, y) to lie within the image range (to allow for the possibility of detecting a face which is slightly less than half visible). However, it obviously must lie either within or just adjacent to the image range to achieve a positive score. So, the parameter search space is "essentially" finite, though very large because it is seven-dimensional.

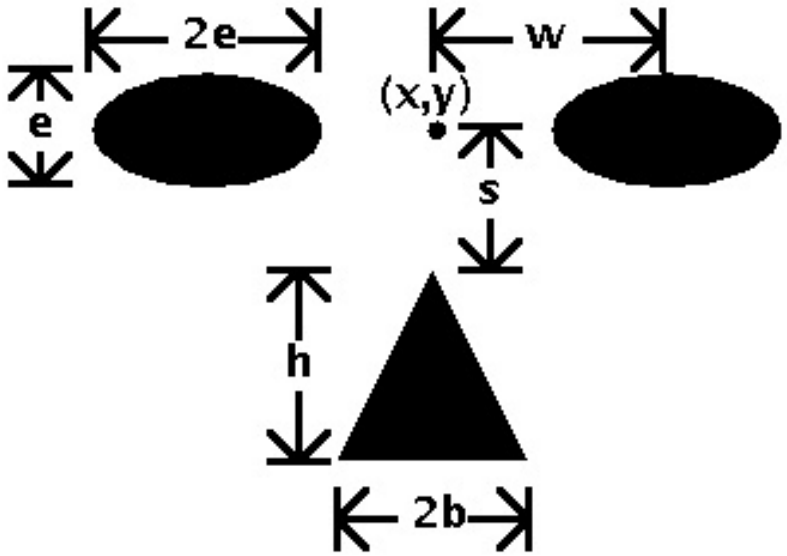

Figure 4. The model parameters of a face (eyes and nose).

Roughly speaking, for the 125×95 image grid used in our experiments, the search space size is on the order of $125 \times 95 \times 20^5/4 \doteq 10^{10}$, about ten billion. (The factor of 4 arises because of the three parameters each restricted to be be less than w.) In any case, for the purposes of this paper, we assume the search space is too large to do a brute-force search of all possible values of all seven parameters.

4. Search Strategies.

To search the large parameter space effectively, we employ a modified Monte Carlo algorithms, as follows.

4.1. Main Algorithm.

Our algorithm begins with a randomly-chosen configuration of the seven parameters, and computes the corresponding score. On subsequent moves, it tries modifying the parameter values, in an effort to find larger scores. Depending on the results of the previous computation, it may use a local update (modifying the previous parameter values just slightly), or a global update (selecting brand new randomly-chosen parameter values, sometimes called a "restart").

Our algorithm accepts various tuning values, including:

- NUMTRIES (int), the total number of parameter configurations whose score functions will be computed. (So, the larger NUMTRIES, the better the result, but the longer the computation will take.)
- NUMLOC (int), the maximum number of successive local updates before the next global update.
- BACKTRACK (boolean), whether moves to lower-score parameter values should be retracted before doing the next local update.
- ONEATTIME (boolean), whether the local updates should modify just a single randomly-chosen parameter (as opposed to all seven parameters).
- ABORTVAL (int), a score value such that any move to any parameter configuration having score lower than this automatically triggers a global update.

That is, a global update is performed at the beginning of the search, and after NUMLOC successive local updates, and after visiting any configuration whose score is lower than ABORTVAL. Otherwise, local updates are performed. For example, setting NUMLOC equal to NUMTRIES, and ABORTVAL equal to $-\infty$, corresponds essentially to doing purely local updates after the first iteration.

One ambiguity should be clarified here. If ABORTVAL equals $-\infty$, and we visit a state with out-of-bounds parameters (corresponding to a score of $-\infty$), then our convention is to always follow this with a global update if the previous update was global (i.e., if a previous global update brought us to the out-of-bounds state), but allow a local update (perhaps after backtracking) if the previous update was local. This rule avoids the problem of a global update to out-of-bounds values, which then wander locally through many different out-of-bounds values without finding any finite scores. Notationally, we denote the corresponding more extreme version (i.e., not triggering a second global update even if the first global update brought us to an out-of-bounds state) as setting ABORTVAL to ∞^*.

In our specific implementation for the experiments below, the global updates set (x, y) to a random point on the image space, and have ranges of about 10 for each of the final five parameters. So, the total number of possible global updates is equal to about $125 \times 95 \times 10^5/4 \doteq 3 \times 10^8$, a fact that will be important in Section 6.

Also, the local updates add an increment chosen from Uniform$\{-2, -1, 0, 1, 2\}$ for x and y, and Uniform$\{-1, 0, 1\}$ for each of the other five parameters; this will also be important in Section 6.

4.2. Simulated Annealing.

For completeness, we also consider a simple version of simulated annealing (e.g. [4]), whereby a local update is retained only with probability $e^{-(\Delta S)^-/T}$, otherwise retracted. Here ΔS is the new score minus the old score, and T is a decreasing "temperature" variable (which we choose to decrease linearly from 10 to 0.1 over the course of the run).

Thus, local updates to higher scores ($\Delta S \geq 0$) are always retained, while local updates to lower scores ($\Delta S < 0$) are only retained with probability $e^{(\Delta S)/T} < 1$ and are otherwise retracted. Here $T \to 0$ corresponds to retracting all moves to lower scores, or equivalently setting BACKTRACK to true. Also $T \to \infty$ corresponds to retaining all local updates, or equivalently setting BACKTRACK to false. Other values of T are partway between these two extremes.

Algebraically, we wish to retract local moves with probability $1 - e^{-(\Delta S)^-/T}$, which corresponds to backtracking if (newscore − prevscore) $< T \log U$, where $U \sim$ Uniform$[0, 1]$ is chosen independently at each iteration.

For fair comparison to the other schemes, we allow the simulated annealing scheme to keep track of the highest score of *all* previously-visited parameter settings, not just the final parameter setting of the run (as is done in "pure" simulated annealing).

4.3. Algorithm Output.

After considering the NUMTRIES different parameter configurations, the algorithm outputs the configuration giving the highest score, together with that score. So, the better the algorithm, the larger (on average) will be the output scores. In the next section, this comparison is used to determine which algorithm settings are best.

5. Experimental Results.

We now describe the results of various experiments with these algorithms, using the Java applet [7] and the experimental framework and test image described previously.

We focus on comparisons of the algorithms with different tuning values. For fair comparison, we fix NUMTRIES at 50,000 for all the trials considered. (Since there are approximately 10^{10} possible configurations, this corresponds to testing less than 1/100,000 of the possible configurations, indicating the scope of the computational challenge involved.) Each algorithm choice considered below was run 20 times on the same image data, to compute a mean score (and the associated standard error), as reported in the following tables.

5.1. Backtracking and Variable Grouping.

We begin by considering the effect of the BACKTRACK and ONEATTIME boolean variables. The following results were obtained:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	1000	True	True	-100	51.6 ± 2.8
50,000	1000	False	True	-100	27.4 ± 2.6
50,000	1000	True	False	-100	30.7 ± 2.9

From these results, it is clear that backtracking improves the algorithm (increasing the average score from 27.4 to 51.6). This indicates that it is wasteful to continue to pursue parameter values which lead to worse estimates in a "random-walk" style of small increments (for related considerations see e.g. [5]).

Perhaps more surprisingly, it is also clear that updating just one randomly-chosen variable at a time is preferable to updating them all at once (increasing the average score from 30.7 to 51.6), even though this results in smaller local moves, presumably because the algorithm is quite unlikely to improve all of the parameter values in a single update. (This is somewhat related to the issue of *optimal scaling* of random-walk Metropolis algorithms, see e.g. [6].)

5.2. Aborting of Local Updates.

Next, we investigate the effect of the ABORTVAL parameter, again running each algorithm 20 times on the same image data. The results are as follows:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	1000	True	True	0	22.5 ± 3.5
50,000	1000	True	True	-50	44.5 ± 2.6
50,000	1000	True	True	-100	51.6 ± 2.8
50,000	1000	True	True	-500	55.1 ± 2.5
50,000	1000	True	True	$-\infty^*$	17.9 ± 3.6
50,000	1000	True	True	$-\infty$	36.5 ± 5.1

These results show a clear trend that as ABORTVAL gets smaller, the average score values *increase*. This suggests that, surprisingly, with these parameters, forcing a global update just because of low score values is not beneficial. It is better to continue with local updates even when in very bad parameter configurations.

On the other hand, if we actually take ABORTVAL to be $-\infty^*$, so that even out-of-bounds parameter values do not trigger a global update, then this leads to very low scores.

This shows that, while local updates are fine even when dealing with poor parameter choices, they are not sufficient when dealing with out-of-bounds parameters which could waste many score evaluations on invalid configurations.

Finally, if we set ABORTVAL to $-\infty$ (so a new global update is triggered if the previous *global* update led to out-of-bounds parameter values, but not if the previous *local* update did), then this is preferable to the $-\infty^*$ option though not as good as setting ABORTVAL to -100 or -500.

5.3. The NUMLOC Parameter, with Finite ABORTVAL.

To investigate the benefits of local versus global updates with a fixed, finite ABORTVAL, we consider different values of the NUMLOC parameter:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	1	True	True	-100	22.7 ± 2.8
50,000	500	True	True	-100	51.6 ± 2.5
50,000	1000	True	True	-100	51.6 ± 2.8
50,000	50,000	True	True	-100	56.5 ± 1.9

These results suggest, again perhaps surprisingly, that large values of NUMLOC only improve the means scores. Even when NUMLOC is equal to NUMTRIES (so no global updates will be triggered by excessive numbers of local updates), the scores are still high (in fact, slightly higher than with more moderate NUMLOC values). As expected, if NUMLOC equals just one, i.e. the algorithm does a global update each time, then this leads to poor results since the algorithm cannot "find" the good parameter values without local searching.

The high scores resulting from large values of NUMLOC may seem surprising. However, even if NUMLOC is very large, there may still be local updates triggered by ABORTVAL, thus partially "nullifying" the effect of large NUMLOC. To consider this, we tried a smaller value of ABORTVAL, too:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	50,000	True	True	-100	56.5 ± 1.9
50,000	50,000	True	True	-500	54.6 ± 2.3

This shows that even if global updates are largely eliminated (due to large NUMLOC and small ABORTVAL parameters), the mean scores are still virtually as good. However, even here, since ABORTVAL is still finite, a global update is triggered whenever out-of-bounds parameters are attempted. We consider this issue next.

5.4. The NUMLOC Parameter, when ABORTVAL equals $-\infty$.

To get a more "pure" measure of the effects of global updates, we do further experiments with ABORTVAL set to $-\infty$ as discussed above, and with various values of NUMLOC. The results are as follows:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	1	True	True	$-\infty$	17.9 ± 2.8
50,000	10	True	True	$-\infty$	36.2 ± 2.2
50,000	50	True	True	$-\infty$	41.9 ± 2.5
50,000	100	True	True	$-\infty$	49.1 ± 3.3
50,000	200	True	True	$-\infty$	49.4 ± 3.6
50,000	300	True	True	$-\infty$	54.2 ± 3.7
50,000	400	True	True	$-\infty$	46.4 ± 4.5
50,000	500	True	True	$-\infty$	45.8 ± 4.8
50,000	600	True	True	$-\infty$	33.6 ± 5.1
50,000	800	True	True	$-\infty$	42.5 ± 5.0
50,000	1000	True	True	$-\infty$	36.5 ± 5.1
50,000	1500	True	True	$-\infty$	19.6 ± 3.5

We see from these results that, with ABORTVAL set to $-\infty$, it is optimal to set NUMLOC to approximately 300, leading to a mean score around 54. The algorithm still performs reasonably well for any values of NUMLOC between about 50 and 800. Values of NUMLOC much smaller (e.g., 10) or much larger (e.g., 1000) than this lead to significantly smaller average score values.

5.5. The Simulated Annealing Option.

As for the simulated annealing algorithm, we recorded the following results:

NUMTRIES	NUMLOC	BACKTRACK	ONEATTIME	ABORTVAL	Mean Score
50,000	1000	S.A.	True	-100	47.2 ± 3.4
50,000	1000	S.A.	True	$-\infty$	28.5 ± 4.3
50,000	300	S.A.	True	$-\infty$	45.5 ± 4.3
50,000	400	S.A.	True	$-\infty$	37.1 ± 4.2

Comparing these results to the corresponding previous results with BACKTRACK set to true, we see that the Simulated Annealing average score values are slightly *lower* than the corresponding pure-backtracking scores. These seems to suggest that, for these search strategies and models at least, there is no benefit to using Simulated Annealing as opposed

to pure backtracking.

Of course, it may be possible to modify the Simulated Annealing algorithm to achieve higher scores. However, we suspect that the real advantage of Simulated Annealing would only become apparent in a different search space having more in the way of steep local maxima which are far from global maxima (a challenging problem that does not really arise in the test images considered here). We plan to consider this issue further in later work.

6. Theoretical Considerations.

From a theoretical point of view, it would of course be desirable to have clear theory about which search strategies and choices are preferable. Of course, such questions are broad and involve many factors. For now, we focus specifically on the case considered in Section 5.4, where we set ABORTVAL to $-\infty$, and BACKTRACK and ONEATTIME both true, and NUMTRIES fixed at the value $N = 50,000$. We then ask, in this configuration, what value L of NUMLOC then maximises the probability of finding very good parameter values?

6.1. An Idealised Theoretical Model.

To put this question in a theoretical framework, we note that parameter values which are fairly *close* to optimal will result in score values which are somewhat close to maximal, or at least somewhat larger than those of randomly-chosen parameter values. Roughly speaking, the score values will fall off linearly as functions of each of the seven parameters, ultimately reaching small (and essentially stochastic) baseline values once the parameter values are far from optimal.

So, we model an idealised version of this question as saying we wish to maximise the function $S(x_1, \ldots, x_d)$ (where $d = 7$ represents the number of parameters) given by:

$$S(x_1, \ldots, x_d) \;=\; \prod_{i=1}^{d} \max\left(0,\; 1 - \frac{|x_i - t_i|}{r_i + 1}\right). \tag{1}$$

Here t_i is the optimal (target) value of parameter i, and r_i represents a "radius" of values that will still give some overlap with the target image, i.e. still have a score which is larger than baseline due to partial overlap with the true image. Thus, if $|x_i - t_i| \geq r_i + 1$ for any i then the score is 0 (i.e., background level), while if $|x_i - t_i| \leq r_i$ then the score is positive (i.e., there is some overlap with the true image) and gets larger as the values x_i get closer to the targets t_i.

Of course, the function (1) is an oversimplification, which does not take into account various local features (such as the possibility of placing the nose where an eye should be).

Still, it is sufficient to provide some theoretical context for the search algorithm optimisations, as we now describe.

6.2. Constraints Based On Global Updates Required.

The function (1) implies that the number of parameter configurations having some overlap with the true image is approximately given by $V \equiv \prod_{i=1}^{d}(2r_i + 1)$. So, if the total number of global update parameter values is G, then the probability that a given global update will result in parameter values having some overlap with the target image is approximately V/G.

Now, if NUMLOC is equal to L, then the number of global updates will be approximately N/L. So, the number of global updates resulting in some overlap with the true image will be Poisson distributed with mean approximately equal to $(N/L)(V/G)$. (Of course, it is also possible that local updates will happen to move the parameter values from those not overlapping the true image, to those overlapping the true image, but this is fairly unlikely and we neglect it from this analysis.)

So, to have high probability of achieving overlap with the true image at least once, we require that $(N/L)(V/G) \gg 1$, or $L \ll NV/G$. This is our first constraint on L.

6.3. Constraints Based On Local Updates Required.

Once a global update provides some overlap with the true image, then the distance to optimal will average about $r_i/2$ in each coordinate. Let c_i be the average net improvement of coordinate i (towards its optimal value) each time it is locally updated. (So, with ONEATTIME and BACKTRACK both true, c_i will be equal to the expected value of the positive part of the proposed increment.) Then each coordinate will need approximately $r_i/2c_i$ local updates to get very close to its optimal value.

Now, over the course of L local updates, the number of times coordinate i is modified will have Poisson distribution with mean L/d. So, to have high probability of converging to optimal parameter values once overlap is achieved, we require that $L/d \gg r_i/2c_i$ for each i, i.e. that $L \gg d \max_i(r_i/2c_i)$. This is our second constraint on L.

6.4. Good and Optimal Values of NUMLOC.

The above discussion indicates that to have high probability of finding good parameter values in this setting, we require that

$$d \max_i(r_i/2c_i) \ \ll \ L \ \ll \ NV/G\,. \tag{2}$$

In particular, we would expect to achieve fairly good results whenever $d\ \max_i(r_i/2c_i) < L < NV/G$, and optimal results when the two ratios are approximately equal, i.e. when

$$\frac{d\ \max_i(r_i/2c_i)}{L}\ \approx\ \frac{L}{NV/G}\,,$$

or

$$L\ =\ L_{opt}\ \approx\ \sqrt{(NV/G)\ d\ \max_i(r_i/2c_i)}\,. \tag{3}$$

We compare these bounds with the previous experimental results in the next section.

As an aside, we also note that (2) can only be satisfied if $d\ \max_i(r_i/2c_i) \ll NV/G$, which requires that

$$N\ \gg\ (G/V)\ d\ \max_i(r_i/2c_i)\,. \tag{4}$$

Equation (4) provides an approximate lower bound on the minimal value of NUMTRIES which, for appropriate choice of NUMLOC, will lead to remotely successful algorithm searches.

6.5. Application to the Test Case.

In our test image described previously, the optimal parameter values appear to be approximately given by: $x = 85$, $y = 62$, $w = 7$, $h = 5$, $s = 6$, $b = 5$, $e = 3$. More importantly, the overlap radii r_i appear to each be approximately 4, i.e. we can adjust each parameter value by approximately 4 while still maintaining some overlap with the true target image.

We then compute that $V = \prod_{i=1}^{d}(2r_i + 1) = 9^7 \doteq 5 \times 10^6$.

Furthermore, in our case, while there are approximately 10^{10} possible configurations, there are only about 3×10^8 possible global updates, i.e. $G \approx 3 \times 10^8$. Thus, $G/V \approx 60$.

As for the c_i, recall that the local increments are Uniform$\{-2, -1, 0, 1, 2\}$ for x and y, and Uniform$\{-1, 0, 1\}$ for the other five parameters. Now, due to backtracking, the net movement of x (say) towards the optimal value is equal to the expected positive part of this increment, i.e.

$$c_1\ =\ (1/5)(0) + (1/5)(0) + (1/5)(0) + (1/5)(1) + (1/5)(2)\ =\ 3/5\,.$$

Similarly $c_2 = 3/5$ and $c_3 = c_4 = c_5 = c_6 = c_7 = 1/3$. We then compute that

$$\max_i(r_i/2c_i)\ =\ \max[4/2(3/5),\ 4/2(1/3)]\ =\ 6\,.$$

Hence, recalling that $d = 7$ and $N = 50,000$, we see that (2) reduces to

$$7(6)\ \ll\ L\ \ll\ (50,000)\,(5 \times 10^6)\,/\,(3 \times 10^8)\,,$$

or

$$42 \ll L \ll 833 .$$

Furthermore, from (3),

$$L_{opt} \approx \sqrt{42 \times 833} \doteq 187 .$$

So, this analysis suggests that the optimal value of L in this case should be on the order of 187, with reasonably good performance for any L between about 42 and 833.

Now, according to the experimental results in the corresponding in table in Section 5.4, the optimal value of L appears to be about 300, with nearly as good results for L equal to 100 or 200, and fairly good results for L between about 50 and 800. Overall, this is quite consistent with the theoretical analysis.

We conclude that, despite the simplicity of our theoretical framework, it provides fairly accurate estimates of the optimal values of NUMLOC in this context.

7. Conclusion.

In this paper, we have presented a simple but useful model for scoring the fit of parameters when searching images for objects such as faces. We have described an interactive Java applet [7] for conducting efficient Monte Carlo searches of the parameter space. We have described a number of different algorithm options such as backtracking, updating the variables one-at-a-time or all together, limiting the number of local updates before the next global update ("restart"), forcing a global update upon reaching a sufficiently low score, etc.

Experiments indicated that backtracking is beneficial (and is not significantly improved further by simulated annealing), and it is best to update the variables in a one-at-a-time fashion. Forcing a global update upon reaching a sufficiently low score is also very useful.

In the absence of such forcing, choosing an appropriate maximum number of consecutive local updates (NUMLOC) is very important. We considered that question in detail through both experimentation and theoretical analysis, and found quite good agreement between the two approaches.

It would obviously be useful to consider similar ideas applied to more complicated images (including real images imported from photographs, see e.g. [1]), to more detailed image models (perhaps involving additional parameters), to more challenging test cases (including those with a stronger multimodal flavour), to less stylised experimental set-ups (e.g. detecting several different objects at once), and to more sophisticated Monte Carlo search algorithms (perhaps involving additional choices and tuning values). We hope to consider some of these issues in future work.

Acknowledgements. I thank Yali Amit and Sven Dickinson for inspiring me to think about these issues.

References

[1] Y. Amit (2002), 2D Object Detection and Recognition: Models, Algorithms, and Networks. MIT Press.

[2] F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, and S. Dickinson (2006), Object Recognition as Many-to-Many Feature Matching. Int. J. Comp. Vision **69(2)**, 203–222.

[3] S. Geman and D. Geman (1984), Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. IEEE Trans. on pattern analysis and machine intelligence **6**, 721–741.

[4] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi (1983), Optimization by simulated annealing. Science **220**, 671–680.

[5] R.M. Neal (1998), Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. In M. I. Jordan (ed.), *Learning in Graphical Models*, 205–225. Kluwer Academic Publishers.

[6] G.O. Roberts and J.S. Rosenthal (2001), Optimal scaling for various Metropolis-Hastings algorithms. Stat. Sci. **16**, 351–367.

[7] J.S. Rosenthal (2008), Computer vision Java applets. Available at:
http://probability.ca/vision