Algorithmic aspects of MAK reconstruction *

M. Hénon
C.N.R.S., Observatoire de Nice, B.P. 229,
06304 Nice, France

November 23, 2004

Exposé au workshop "Large-scale reconstruction", August 30 – September 3, 2004, Nice.

1 Introduction

As has been explained by UF, in the MAK approach, the reconstruction problem can be reduced to the following form: we are given N initial points $\mathbf{q}_j$ and N final points $\mathbf{x}_i$, and we want to find a one-to-one correspondence between these points, defined by [a function] $j(i)$, which minimizes the sum of the squares of the distances:

$$\sum_{i=1}^{N} |\mathbf{x}_i - \mathbf{q}_{j(i)}|^2. \qquad (1)$$

*Fichier `cosmo/Workshops/workshop04.tex`

This is a classical optimization problem, known as the *assignment problem*. The general form of the problem is to minimize a quantity

$$\sum_{i=1}^{N} c_{ij(i)} \tag{2}$$

where c_{ij} is a given *cost matrix*.

I will report on the work done here in Nice in relation to this assignment problem. The main objective was to develop a program sufficiently fast to be able to treat a large number of objects [; "large" being as much as one million or more]. It is known that assignment problems can be solved in a time of order N^3; this, however, is not fast enough for our problem.

The actual work of programming and testing was done essentially by 3 people: Roya Mohayaee, Andrei Sobolevskii, and me †, but we have benefitted from discussions with a number of colleagues. title page

2 Testing the algorithms

A number of algorithms were tried. They were tested on two different categories of problems. First, we ran N-body simulations, [typically with $N = 2$ million bodies,] starting with the objects uniformly distributed in a volume. We selected a subset of these initial objects, and we computed the optimal assignment between them and the corresponding final positions. [Incidentally, this was also a test of the MAK method.]

The second category of problems came from real catalogs of galaxies. In that case, the initial positions are of course un known, and they were assumed to be uniformly distributed.

3 Bertsekas' auction algorithm

The first algorithms which we tried were too slow. But progressively we discovered and adapted faster algorithms. I will concentrate on the fastest algorithm which we have found. This is the *auction algorithm*, invented by Dimitri Bertsekas.

[There are several ways to describe this algorithm; here I choose one which follows closely the actual program.]

The x_i are considered as *persons*, and the q_j as *objects*; and the persons compete for the objects in an auction-like fashion.

First, we define the assignment problem in terms of *benefits* instead of costs: we wish to *maximize* a quantity

$$\sum_{i=1}^{N} b_{ij} \tag{3}$$

where b_{ij} is a given *benefit matrix*. We pass from one formulation to the other simply by taking

$$b_{ij} = -c_{ij} \tag{4}$$

or, if you prefer to work with positive numbers:

$$b_{ij} = C - c_{ij} \tag{5}$$

where C is a large number. [These two formulations of the assignment problem are a frequent source of confusion.]

We assume now that each object j has a *price* p_j. This price varies during the course of the computation. Then we define the *value* of object j for person i as

$$b_{ij} - p_j. \tag{6}$$

Now we try to fix the prices and to assign the persons to objects (i.e. determine a one-to-one correspondence) in such a way that each person is assigned to the object with maximal value for that person, i.e.

$$b_{ij_i} - p_{j_i} = \max_j (b_{ij} - p_j). \tag{7}$$

If we succeed in doing this, we say that we have achived *economic equilibrium.* Now a remarkable property can be proved: this equilibrium solves the assignment problem for the benefits b_{ij}.

3.1 Naive auction algorithm

A first algorithm for finding an equilibrium is called the *naive auction algorithm.*

The computation proceeds in *rounds.* At the beginning of a round, there is a set of prices, and a partial assignment: some persons are assigned to objects which verify (7), while other persons are unassigned. Then the round proceeds as follows:

1. We select one unassigned person i.
2. We find the object j_i which offers maximum value, and we assign person i to object j_i.
3. If another person was assigned to j_i, it becomes unassigned.
4. We now increase the price of object j_i by the largest possible amount such that it still offers the best value to person i. (This in order to discourage competition!).

This is done as follows: we consider the best object value:

$$v_i = \max_j (b_{ij} - p_j) \tag{8}$$

which occurs for $j = j_i$, and we determine also the second best object value:

$$w_i = \max_{j \neq j_i} (b_{ij} - p_j). \tag{9}$$

Finally, we replace the present price p_{j_i} by

$$p_{j_i} + v_i - w_i. \tag{10}$$

This is continued until all persons are assigned.

3.2 Auction algorithm

Unfortanely, this simple algorithm sometimes gets stuck. This happens if more than one object offers best value. Then the price does not change and the computation never ends.

To avoid this, we define a [small] parameter ϵ, and instead of (10) we define the new price by

$$p_{j_i} + v_i - w_i + \epsilon. \tag{11}$$

It can then be shown that the computation ends always in a finite number of rounds. However, what we obtain is not the exact economic equilibrium: the condition (7) may be verified only within ϵ. If ϵ is small, it can be shown that the end result is in a sense close to the economic equilibrium. In particular, an important result can be proved: *If the benefits b_{ij} are integer, and if*

$$\epsilon < \frac{1}{N} \tag{12}$$

then we do reach the economic equilibrium, and the problem is solved.

3.3 ϵ-scaling

If we start with a small ϵ, the computation is likely to take a long time. The program does not quite get stuck, but it proceeds by very small steps, and the computation time tends to be proportional to $1/\epsilon$.

To avoid this, we introduce a final refinement: ϵ-scaling. The program executes a succession of *phases*, with progressively decreasing values of ϵ, until a last value satisfying (12). The initial value of ϵ can be quite large; experience suggests that it should be of the same order as the benefits.

The successive values of ϵ are controlled by a few parameters. Experience shows that these parameters should be carefully adjusted for best results.

4 Sparse version

So far I have described the *dense* version of Bertsekas' algorithm, in which all assignments are a priori allowed. However, the algorithm is naturally adapted to the case of *sparse* problems, in which only a subset of the (i, j) pairings are allowed. In our case, it is natural to suppose that a x_i will be paired with a q_j only if their distance is comparatively small. Thus, we define a *critical distance* d_{crit}, and we allow only pairings with a distance less than d_{crit}.

One practical problem with the sparse version is that we have in principle to compute and store in advance the allowed pairings. This requires a table with dimensions $O(N^2)$, which might be too large.

This table can be avoided if the initial positions lie on a

cubic lattice; more precisely, if the initial positions are all the vertices of a cubic lattice lying inside a simple surface, such as a sphere or a cube. In that case, for a given final point $\mathbf{x}_i$, the list of the allowed pairings can be easily generated at run time. Fortunately, this is a case frequently encountered, since the initial distribution of mass is usually assumed to be homogeneous.

The figure † shows some results. It is a log-log plot; the horizontal axis is N, number of objects, and the vertical axis is computing time divided by N^3. The upper points correspond to an algorithm of Burkard and Derigs, which was our second best [on rimsky]. The next points below are for the dense auction algorithm; it can be seen to be about 10 times faster than BD. Finally, the lower points are for the sparse version. The improvement in computing time is spectacular: about a factor 50! fig5

[Using the sparse version, a computation with $N = 10^6$ could be done in 17 days [on debussy 1]].

However, a problem is that the choice of the critical distance is quite delicate. If we take it too small, we may miss the optimal assignment. (In fact, if it is really too small, the program may be unable to find an assignment at all.) If the critical distance is too large, the computing time increases rapidly and we lose the advantage of the sparse algorithm.

5 Constant volume and constant density

I mention a last point. These computations were done at *constant volume*: the initial points are selected randomly in a

given volume. It can be seen that the asymptotic behaviour of the sparse algorithm is approximately in N^3.

However, one can also proceed in a different way, and keep the *density* constant when N is increased: for smaller N, the points are selected from a smaller volume, in such a way that the density remains the same. The results are then quite different: the time dependence for large N appears to be of the order of $N^{1.5}$.

I stop here; but this afternoon, Andrei will continue on the same subject and describe refinements of the auction algorithm adapted to the case of real catalogs of galaxies.