Application of Artificial Intelligence Techniques to Economic Planning

PAUL COCKSHOTT
Department of Computer Science, Strathclyde University, Glasgow, Scotland

Abstract

The relevance of computer science to economic planning is defended. An algorithm for constructing a balanced economic plan is presented and found to be of time order $Nlog(N)$ in the complexity of the economy. The time taken to perform a complete plan optimization in natural units for a whole economy is estimated.

Plans and computational costs

This is a paper by a computer scientist that disputes an economic hypothesis. The hypothesis rests upon certain premises about computation and is therefore open to criticism from outside economics. The hypothesis is that: the complexities involved with performing the calculations required to optimize an economic plan in natural units are so great that they are beyond the realms of feasibility and that in consequence all rational economic calculation must proceed by means of the intermediary of prices. The hypothesis is an old one. It was originally proposed by Von Mises (1936) between the wars. In recent years it has been ably restated by Nove (1983) and has gained circumstantial support from recent developments in the USSR. It is argued here that computer science has something relevant to say on the topic. It is then argued that the difficulties experienced in socialist economies with planning in natural units are partly a consequence of the particular formalism used to express the problem. Finally, it is argued that techniques developed in artificial intelligence can be applied to solve planning problems with economically acceptable computational costs.

In what follows it is assumed that planning authorities start out with a pre-given objective in terms of net output. That is to say they wish to achieve a specified bundle of output flows at the end of the plan period. If possible they would like this bundle of outputs to be exceeded, but excesses of some commodities are not desired if they result in shortfalls in others. In other words these are the classic assumptions proposed by Kantorovich (Ellman 1972). We do not concern ourselves with how these plan requirements are arrived at. It is further assumed that the plan authorities have effective property rights over all means of production and can allocate them between different productive activities in order to achieve plan goals.

Relevance of computer science

Computation is always a physical process. It is always performed by real physical mechanisms. These may be humans, humans aided by pen and paper, humans aided by calculators or electronic computers. At some point in the future these may be replaced

by other physical mechanisms, perhaps based on optics. Whatever the mechanism, it has an economic cost. Human statisticians must be paid, computers must be built.

There exists a body of laws which describe the costs of computation (Kronso 1987). The investigation of these laws is the task of computer science. In conjunction with the disciplines of electronic engineering and software engineering, it develops practical techniques for the solution of large-scale computations. The computational feasibility of a problem depends upon the rate at which the number of elementary arithmetic operations required to solve it grows with the size of the problem. If we label the size of the problem N then the complexity of a problem is characterized by some function

$$complexity = F(N)$$

which defines the least upper bound of the number of elementary arithmetic steps required. Tractable problems should have polynomial complexity functions. For really large N it is desirable to have a function of linear or log.linear complexity. These costs are conventionally expressed in terms of time. They may alternatively be mapped into costs in terms of space. By physical replication of components it is often possible to reduce the time taken to perform a computation, but the product of time by space occupied tends to be invariant for a given algorithmic technique and a given problem. This has obvious economic implications. The time and space abstractions of complexity theory translate into real economic costs. A computation is not worth doing if the answer arrives too late to do anything about it. If the cost of the computer required to solve a problem is greater than the savings to be made from solving it, it is better not to try.

A large number of problems may be viewed from the standpoint of computation. We can for instance consider the operation of a market economy as a computational process. Loosely speaking we would describe the 'problem' as being defined by the available physical resources and the demand schedules of the consumers. The objective of the calculation would be to 'arrive at a set of prices and a distribution of resources that optimally met the demand schedules. We are justified in thinking of this as a computational process because of a very powerful theorem of computer science that any finite physical process may be viewed as a computation and simulated on a computer (Deutsch 1985) with an appropriate program. A market economy is a rather slow computer since the basic steps of information transformation (price changes) only come about by the intermediary of changes in the physical volume of outputs. The elementary steps in the computation may take months or years, during which the availability of resources and demand schedules change. As a result the computation does not terminate.

A planning bureau in a centrally controlled economy is more obviously a computational process. In this case the computation is not tied to alterations in the volume of physical output but proceeds either through the exchange of draft plan proposals between economists (Kornai 1975) or through the execution of programs on the planning bureau's computers. In currently existing planning bureaus a considerable part of the process is still human mediated, which slows down the computational cycle. It may be the case that the speed to compute a plan in this way will actually be slower than the relaxation time of a market. Computer technology has delivered very big increases in productivity over the last forty years. The speed difference between hand calculation and doing the same thing on the fastest modern computer is about 10 to the power of 11. No other technology has achieved increases remotely like this. This raises the possibility that an entirely automatic computer program could perform the computations

necessary for the control and balancing of production far faster than either a market or a planning bureau. To demonstrate that this is feasible we have to show that the problem of plan allocation can be cast in a form that is amenable to computer solution and that the complexity function of this computation has a time/space product that is economically acceptable.

Limits to the formalism of linear programming

Kantorovich demonstrated that the plan problem as formalized by him was logically soluble using linear programming techniques. Although it is logically possible to compute the correct allocation of resources to industries by these techniques, their practical application is hindered by several factors. Among these are the lack of data or poor quality of data available to the planning authorities in socialist economies and the technical backwardness of their computing machines. More significant is the question of computational complexity. Nove emphasises the scale of the problem, saying that there are 12 million distinct products in the Soviet Economy. He quotes a Soviet Economist as saying that it would take the whole population of the world millions of years just to solve the equations required for the plan of the Ukraine.

The cost of solving linear programming problems grows non-linearly with the number of industries considered. Just to store the technical coefficients as an input/output matrix for the USSR economy would take around 1000000000000000 bytes of computer memory. At current prices of around $1000 per million bytes, this means the computer would cost upwards of 100 billion dollars. This alone would rule out applying a linear program to the whole economy even before we consider the running time of the program.

To be acceptable the computation period should not exceed a few months, otherwise decisions arrive too late. Ideally we would like answers the same day. The cost of the computers and communications networks needed for the process should be less than the existing computing budget of an advanced economy, so that the computational tail does not wag the economic dog. We next argue that the problem of creating a balanced plan is order $NlogN$ and computationally tractable provided that it is cast in terms of a different optimization model.

Representing the problem

The approach is to construct an internal computer model of the complete production structure of the economy and of the desired pattern of output. A form of search algorithm is then undertaken to discover a pattern of resource allocation that is close to optimal. It only gets close to optimal since the type of search procedure used is an iterative optimization which is terminated once an acceptable level of performance is achieved. The production structure of an economy is conventionally represented as an Input/Output matrix from a computational viewpoint. The memory storage requirement of a matrix grows as

$$N^2$$

and the time order of matrix operations is greater than linear. Advantage is taken of the fact that real input/output matrices will, if expressed in natural rather than value units, be sparse. This allows the problem to be remodeled. Assume that there exists an enumerated type *PRODUCT* in our computational model of the economy such that

the range of values of the type corresponds to the range of real products in the economy. An implementation of the type might be the bar-code number associated with each product. The other types used in the model are *STOCKS, FLOWS, TECHNOLOGIES*, and *INDUSTRIES*. A STOCK is defined to be an ordered pair of type (integer, PRODUCT) defining a number of units of a product. A FLOW is also of type (integer, PRODUCT) but is defined to be of dimension

$$\frac{d}{dt}(STOCK)$$

By convention we define the consumption of a product to be a negative flow and the production of a product to be a positive flow (having negative and positive valued integer parts, respectively).

A TECHNOLOGY is defined to be a function of type (*STOCK → *FLOW). That is to say it maps a set of stocks to a set of flows. (In what follows the notation *X will mean the type of a set of X.) The interpretation of this is that the technology will allow a production process to take place such that: a given set of stocks will cause a net consumption of some products and a net production of others. Specifically, we assume that to generate a given net output, stocks of inputs must be combined in fixed proportions. So that:

$$STOCK_j = Ic_j$$

Where I is the intensity with which a technology operates,

$$c_j$$

is a constant, and

$$STOCK_j$$

is the minimum stock of input j needed to attain this intensity. It is assumed that the flows induced by the technology will be of the form:

$$FLOW_j = f_j I$$

where the f are constants. An INDUSTRY is characterized by the combination of a set of stocks with a technology, hence (*STOCK, TECHNOLOGY).[1] The industry's dynamic behaviour is characterized by the application of the technology function to its stocks.

The above representation of the problem has the great advantage over linear programming approaches that it involves no matrices. In practice the matrix of technical coefficients of the economy would be very sparse. By using a set representation, the same information can be encoded much more efficiently. Using a suitable compact set representation the store required will grow proportionally to the product of the number of types of goods times the number of direct inputs that go into each distinct good. Because the mean number of direct inputs to a product is likely to be hundreds not millions, the memory costs for a representation of an economy are reduced by several orders of magnitude. A computer of the appropriate size would be expected to cost a few million dollars rather than hundreds of billions of dollars.

[1] In the algorithm it is assumed that there is only one technology associated with each product and that there is no joint production. We are investigating the more general case with multiple alternative technologies.

The plan problem

The plan problem can be defined as follows: given a set of stocks that exist at the current time period, and given a desired pattern of consumption of consumer goods, and a pre-given set of technologies, find the industrial structure that best meets this. This involves deciding how to allocate the aggregate stock of means of production between all of the industries.

This can be solved by using techniques borrowed Irom artificial intelligence. Welfare economics is dependent upon the assumption that consumers are capable of chosing an optimal consumption pattern subject to certain constraints. This is a particular representation within the domain of economics of the ability of neural systems-human brains-to perform constraint satisfaction computations. Humans carry out constraint satisfaction computations all the time with our most basic physical movements. When we walk across the room and pick something up, our brain has solved an enormously complex constrained cost minimization function that has as its parameters all sorts of information about the degrees of freedom of our joints, the lengths of our bones, the impossibility of walking through tables, the fact that energy consumption is minimized by walking on our feet rather than our knees etc. We are unaware of them because trial and error during infancy specialized our brains for this sort of calculation.

Economic planning is a problem of constraint satisfaction. Neural systems are consumately effective at constraint satisfaction, so it is beneficial to apply what has been learned through the study of neural networks to this area. Neural nets can be thought of as collections of entities with local interactions. The same can be thought of industries. An industry interacts with its immediate suppliers and customers. A neurone interacts with the other neurones that supply it with input signals and in turn drives output signals to other neurones. The intensity with which an industry is operated can be modeled by the frequency with which a neurone fires. A real neural analogue computer might have a neurone to represent each industry and would be set up with appropriate weights on its synapses to represent the strength of its coupling to other industries. The system is then presented with external stimuli representing the desired pattern of output and the available inputs and is 'trained' to select a pattern of industry activation that meets these constraints. In practice we would simulate the neural analogue computer on one or more digital computers. We end up with a digital computer simulating a neural computer simulating the total production function of a whole economy. But the principle of training with positive and negative reinforcement remains.

In order to achieve this we introduce function which we term a Harmony function. This is loosely based upon the notion of Harmony used in the literature on neural nets (Smolensky 1986). The notion behind it is that Harmony is a real-valued function that measures how closely the net output of the economy corresponds with the goal. The function TotalHarmony(output,goal) where

$$output, goal : *FLOW$$

may be evaluated by summing the contributions to TotalHarmony from each product. We define the function PartialHarmony(p) where

$$p : PRODUCT$$

to take on the value 0 when the output of a product exactly corresponds to the goal; it becomes steeply negative as output falls below the goal and becomes slightly positive

when output exceeds the goal. This corresponds to the notion that shortfalls are more important than surpluses. A possible form of the partial harmony function would be:

$$PartialHarmony(p) = H(scale(output(p), goal(p))) \tag{1}$$

Where the scale function is of the form:

$$scale(o,g) = \frac{o-g}{g} \tag{2}$$

and the function H takes the form:

$$H(x) = \begin{cases} \frac{1}{2} & \text{if } x > 0 \\ -x^2 & \text{if } x \leq 0 \end{cases} \tag{3}$$

Since this function has a downwards sloping first derivative it mimics the economists' notion of diminishing marginal utility. The partial harmony function depends upon relatively local information: the computed supply and demand for the product of an individual industry. This makes it suitable for use in a neural-motivated model.

Given the partial harmony function we can construct a total harmony function:

$$TotalHarmony = \Sigma_p PartialHarmony(p) \tag{4}$$

We redefine the problem as that of finding an algorithm that will adjust the distribution of stocks between industries so as to maximize harmony.

The algorithm

We start off with a random distribution of stocks between all industries, subject only to the constraint that stocks of a product are only allocated to those industries that use it as an input.

1. Find the rate-limiting factors

For each industry determine the product for which the input stock acts as a rate limiting factor. Assume that the production function for the industry in question requires that the inputs must be combined in fixed proportions. This step will be of order

$$k_0 NM$$

where N is the number of industries and Mis the mean number of inputs per industry.

2. Remove non-critical resources

If we have determined the critical resource for a production process and if we have a linear production function we can determine the stock of each other product that is required to optimally match the stock of the current critical resource. This is again subject to the assumption that the inputs must be combined in fixed proportions. We call this the balancing stock. Given the balancing stock of each non-critical input we can deduct any excess stocks and assign them to a global reserve. This step will again be of time order

$$k_1 NM$$

This step does not reduce net production as the resources moved to the central reserve are defined to be non-essential. In consequence, total harmony is not reduced by this step.

3. Compute partial harmonies

Evaluate the partial harmony of each product. This involves calculating the net production of each product, comparing it with the goal and applying the harmony function. If this is done by iterating through each industry and evaluating the product flow contributed by that industry the time order of this will be

$$k_2NM + hN$$

where h is the cost of applying the partial harmony function to a single product.

4. Compute mean harmony

Given the partial harmonies, the mean and total harmony can then be computed. This will be of order N.

5. Sort in order of harmony

We assume that there is only one industry acting as a net producer of each product. The harmony function originally applies to products; we now associate each industry with the partial harmony of its product. This enables us to order the industries in terms of ascending harmony. As a sorting operation this will be of complexity $Nlog(N)$.

6. Reallocate reserves

The stocks in the global pool are reallocated to industries starting with those industries that are least harmonious. (Note that these are purely notional transfers performed on the representation of the economy in the computer; no real transfers occur until the whole computation has terminated.) For each of these industries we calculate the additional stocks required to bring the industry up to mean harmony and allocate these to it from the global pool. The time order of this stage will be

$$k_3NMp$$

where p is the proportion of industries that can have this done to them before stocks run out. As each industry has resources allocated to it, it is moved into the appropriate position in the list of industries and the mean harmony is re-evaluated. The cost of this operation will be of order $pN\ log\,(N)$.

7. Reduce harmony peaks

Up to this point all steps have tended to conserve or increase harmony. This is because they all tend to maintain or increase total production. We now have to alter the composition of production towards the most harmonious overall structure. This involves reassigning resources from those industries with the highest harmony to those with the lowest. Since the derivative of our harmony function

$$\frac{dH}{dx}$$

decreases throughout its range, the system is characterized by diminishing marginal harmony. In consequence, total harmony can in some circumstances be increased by moving resources from the production of products with above-average harmony to those with below-average harmony. Our next step is to transfer resources from the most harmonious to the least harmonious branches of production.

The set of products that are of above-average harmony is identified, the outputs of the industries producing them are scaled down until they are producing at average harmony, and the resources released are allocated to the global reserve. The complexity of this operation is

$$k_4NMq$$

where q is the proportion of products of above-average harmony.

8. Iterate steps 6 and 7 till increase in harmony is small

The crucial point here is how often the process has to be iterated. The limit to the complexity of the whole operation will be:

$$R\left(pN\left(k_3M + log\left(N\right)\right) + k_4NMq\right)$$

where R is the number of iterations required. If we assume that the number of products in an economy is of the order of a million then M may well be greater than $log\left(N\right)$. If we assume that M is of the order of 100 then the number of steps for the balancing of a million-product economy would be of the order of

$$Rk_5 10^8$$

For an optimized program we might estimate the number of steps to be between 10 billion and 100 billion. Given that the fastest current computers operate at several billion operations per second (Frenkel 1986), this seems to be well within the bounds of feasible computation.

Experimental verification

The algorithm was programmed in the C programming language and a series of experimental runs made with simulated economies. The inputs to the program were:
(1) a set of N technologies,
(2) a set of target outputs for each product,
(3) a set of stocks of means of production.

The inputs were prepared by another program that ensured that the technologies were feasible, i.e. that the Sraffaian (Sraffa 1960) basic sector was capable of producing a surplus product, and that sufficient stocks of means of production were provided to meet the goals. The particular details of the technologies, targets, and stocks were, subject to these constraints, produced by a random-number generator.

It was observed that the algorithm as given above did redistribute the stocks between industries in order to equalize harmony levels between industries. However it was found that industries converged upon a mean level of harmony that still left unused stocks of resources.

There seem to be two alternative interpretations of this tendency to leave excess stocks. One possibility is that the system gets trapped in a local maximum of harmony

that is below the global optimum such that no small variation in resource allocation would allow the system to escape from this local maximum. Alternatively, the fault may lie with the algorithm having an excessive tendency to converge towards the current mean harmony level. The problem of local maxima is also encountered in neural net simulations and it is avoided by using the technique of simulated annealing (Kirkpatrick *et al* 1983). In that case, thermal noise is added to ensure that the system moves towards a global maximum of harmony. The algorithm was thus modified to incorporate simulated annealing.

In steps 6 and 7 a target output is computed for each industry such that production at this level would result in the industry being at mean 'harmony':

$$target_i = H(meanharmony) \tag{5}$$

The target is computed using the inverse harmony function H' for the industry concerned.

In order to overcome the strong convergence on the mean and the possibility of local maxima induced by this formula, an amplification a and a random noise variable n were added so that the output level was biased upwards:

$$target_i = (1 + n + a)H(meanharmony) \tag{6}$$

This also should allow for the system to escape local maxima. With each successive iteration the bias a and the noise variable n were reduced, allowing the system to go through two phases. In the first phase the target is dominated by the amplification bias, and all industries increase their outputs until resource constraints inhibit this. During the cooling phase the amplification bias tends towards zero and resources are gradually redistributed between industries. Monte Carlo type tests were performed on three versions of the algorithm: version 1 did not use amplification or thermal noise, version 2 used amplification alone, version 3 used both amplification and noise. A total of 49 runs of each the three algorithms were made. In all 49 runs the number of industries, the output goals, the technology, and the available stocks of resources were held constant. Each run used a different initial allocation of these resources between industries. For each of these initial allocation patterns the three versions of the algorithm attempted to find a maximally harmonious final resource allocation. The mean and standard deviations of the harmonies were then recorded for each algorithm on each run (Cottrell 1989). The results are summarized in Table 1. These seem to show that there is a statistically significant difference between version 1 and versions 2 and 3. The 95% confidence intervals for mean harmony are non-overlapping. On the other hand, there is not a significant difference between versions 2 and 3. Although the average mean harmony is a little higher when thermal noise is added to amplification, the 95% intervals for the populations are substantially overlapping. This implies that we should fail to reject the null hypothesis of equality between the two population means. The conclusion is that the addition of thermal noise is not worthwhile.

Table 1. Monte Carlo test results

Algorithm		
Ver.sion 1 Version 2 Version 3 Average mean harmony -0..9473 1.3131 1.3180 Standard error 0..0012 0.0408 0.0406 Top of 95% confidence interval -0..9498 1.2314 1.2367 Bottom of 95% confidence interval 0..9448 1.3948 1.3994		

Verifying that solution is correct

Does the algorithm return the same solution as would have been arrived at by analytic means? In order to determine this, it was set the problem of computing the maximal harmony resource allocation for a system for which there was a known analytic solution. The approach was to define a set of goals and a set of technologies to achieve these goals, and then to analytically determine the set of resources that were just sufficient to meet these goals with the given technologies.

Let F be the input output flow matrix, then the net production flow matrix P is defined by

$$P = (I - F)$$

Now let the matrix of capital stocks required to sustain one unit of production for each industry be denoted by C and the goal vector by g. We can obtain the vector of stocks s just sufficient to meet the goals from the equation:

$$s = C \cdot (P^{-1} \cdot g)$$

If this quantity of stocks is harmoniously allocated between industries then the mean harmony of the system should be zero. This follows from the definition of harmony, which states that it is zero when outputs exactly equal goals. When the plan-balancing algorithm was presented with a collection of industries whose total stocks had been calculated in this way, it terminated with a mean harmony of -0.0089. Given that the analytic solution assumed a real-valued stock vector which was rounded down

to integer form for the plan-balancing algorithm, this was taken as evidence that the solution produced was correct to within rounding errors.

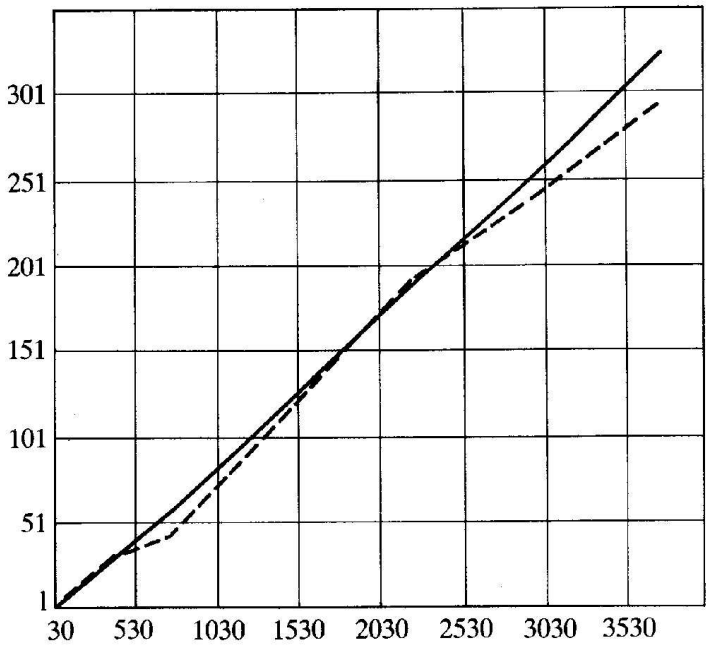

Fig. 1 Run time against number of industries
X axis = number of industries
Y axis = run time in seconds
Solid line M = 13, dotted line M = 25

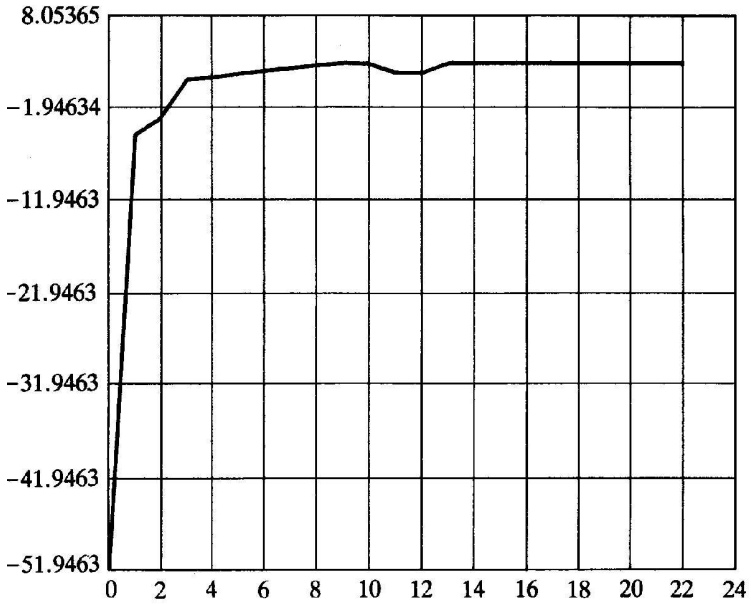

Fig. 2 Harmony on successive iterations
X axis = iteration numbers
Y axis = harmony

Experimentally determined time order

Test runs were done with various numbers of industries. At the lower limit the number of industries was 30, at the upper limit 3750. In Fig. 1 two plots are shown of the computation time against number of industries for systems with $M = 13$ and $M = 25$, respectively. It will be observed that the run times are approximately a linear function of the number of industries. In general it was found that systems with large M converged after slightly fewer iterations than systems with small M and that for a given value of M the number of iterations was relatively independent of N. Figure 2 shows the evolution of mean harmony with successive iterations. The two phase development: rapid expansion followed by equilibration can be clearly seen.

Conclusion

The experimental results confirm the initial complexity analysis of the algorithm. The computer used for the computation had a floating-point arithmetic performance of less than 1 million operations per second. It was able to handle a system of 3705 industries in just over 320 seconds. It seems reasonable to project a similar compute time for balancing a plan of an entire economy on a modern super-computer. Nove gives an estimate of 12 million distinct products in the economy of one of the super-powers. This is an increase in the scale of the problem of about 3 orders of magnitude as compared to the experiment. The latest supercomputers have a throughput of several billion operations per second. This is again a 3-orders of magnitude improvement. Because

the algorithm depends upon local information, it should be suitable for multiprocessors. This implies that plan balancing in natural units is approaching the limits of what can be practically computed. Since computer technology advances quickly, what is at present marginally possible will soon be routinely possible. Such computations would only be as valid as the data available. To work they would presuppose the existence of an automatic data collection network, which relayed up-to-date information on partial production functions to the computer that performed the optimizations. We have argued elsewhere (Cockshott and Cottrell 1989) that this is well within the capabilities of current microcomputer and telecoms technology. We conclude that automated resource allocation by computer constitutes a third economic alternative to market allocation or bureaucratic allocation.

References

COCKSHOTT W. P. and COTTRELL A. (1989). 'Labour values and socialist economic calculation', Economy and Society, in press.

COTTRELL A. (1989). 'Analysis of the Monte Carlo test of the planbalancing algorithm', private communication.

DEUTSCH D. (1985). 'Quantum theory, the Church-Turing principle and the universal quantum computer', Proc. R. Soc. Lond. A400 97-117.

ELLMAN MICHAEL. (1972). Soviet planning today (Cambridge University Press, Cambridge).

FRENKEL KAREN A. (1986). 'Evaluating two massively parallel machines', Communications of the ACM 9 (August).

KIRKPATRICK S. JR., GELATT C. D. and VECCHI M. (1983). 'Optimization by simulated annealing', Science 220, 671-680.

KORNAI F. (1975). Mathematical planning of structural decisions (Akademiai Kiado, Budapest).

KRONSO LYDIA. (1987). Algorithms: their complexity and efficiency (Wiley).

MISES L. VON. (1936). Socialism: an economic and sociological analysis (Jonathan Cape, London, trans. J. Kahane).

NOVE ALEC. (1983). Economics of feasible socialism (Allen & Unwin).

SMOLENSKY P. (1986). 'Information processing in dynamical systems: foundations of harmony theory', in Rumelhart, David, Parallel distributed processing, vol. 1 (MIT Press).

SRAFFA P. (1960). Production of commodities by means of commodities (Cambridge University Press, Cambridge).