# THE USE OF THE HARMONY ALGORITHM FOR MULTI-YEAR ECONOMIC PLANS

PAUL COCKSHOTT

Abstract. The use of linear optimisation for the construction of economic plans was first introduced by Kantorovich[1, 2] with an algorithm similar to those later developed in the West by Danzig[3, 4]. In recent years linear optimisation packages have become readily available on desktop computers, an example being the lp-solve system[5]. In this paper I first show how the lp-solve package can be used to construct macro-economic plans starting from information in IO table. I then examine the empirical computational complexity of the package in dealing with this sort of problem. This will show that the complexity is too great to allow the package to be applied to highly disaggregated IO tables.

As an alternative to lp-solve I explain the Harmony Algorithm[6] and how that can be extended to the problem of multi-year plans. Performance measurements are given which indicated that the Harmony algorithm has a markedly lower computational complexity than lp-solve, making it more suitable for highly disaggregated plans.

## 1. The plan problem

Kantorovich defines the plan problem as to find a way to combine a finite number of techniques along with a pre-given vector of resources in order to maximise the fullfillment of a plan target. The target itself was specified in terms of a vector that specifies the mix of outputs to be produced. For example a plant producing car engine parts might have to maximize the output supplied in the fixed proportions N engine block, 4N pistons plus N crank shafts. A technique, in his terms, was a linear combination of resources that produced an output in fixed proportions. Given appropriate data about sales of consumer goods and planned state purchases of public goods, the entire operation of the productive economy could be defined in these terms.

Given such a plan specification, the Kantorovich's algorithm gave the minimum resource using combination of techniques. The answer provided by his algorithm was the relative intensities with which each technique should be used.

During the 1960s when Kantorovich was working the problem of macro-economic planning, computing resources were far less developed than today, and the preparation of detailed plans for a whole economy using his techniques was not possible. More recently the entrepreneur Jack Ma who founded Alibaba has revived the idea[7], claiming that with big data, networks and modern computing it should be possible to have detailed real-time planning of the Chinese economy. For this to be feasible in an economy as big as China, the computational complexity of the calculations have to be tractable.

What do we mean by tractable in this context?

Normally we treat an algorithm as tractable if it is of polynomial order, but when dealing with very large datasets, we require the stronger constraint that the run time be bounded by a low order polynomial. Unfortunately, that does not appear

*Date*: 28-8-18.

to be the case with the linear programming algorithms descended from the work of Kantrovich.

## 2. Performance of lp-solve for economic planning

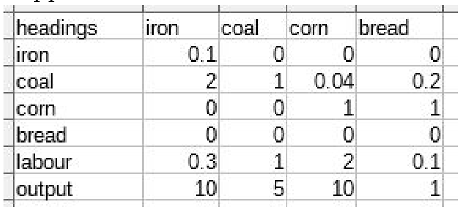
As a test we produced an macro-economic planning front end to the widely used lp-solve package. The package[1] allows one to experiment with macro-economic planning. Using it one can compute multi-year plans for either toy economies, or in principle, it can be applied to compute sectoral plans for whole economies using input output tables. It is intended to be run in a Linux environment, from the command line. It takes data in spreadsheet form allowing published national IO tables to be used as the starting point for feasability studies.

It is invoked thus:

```
java planning.nyearplan flow.csv cap.csv dep.csv targ.csv > plan.lp
```

The spreadsheets are supplied in comma separated value (.csv) form, a widely used exchange format. First comes a flow input/output table in standard column format called, in the example, `flows.csv`.

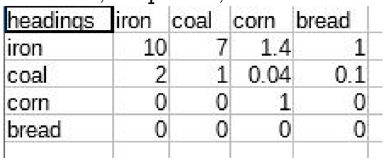
Suppose this contains



| headings | iron | coal | corn | bread |
|---|---|---|---|---|
| iron | 0.1 | 0 | 0 | 0 |
| coal | 2 | 1 | 0.04 | 0.2 |
| corn | 0 | 0 | 1 | 1 |
| bread | 0 | 0 | 0 | 0 |
| labour | 0.3 | 1 | 2 | 0.1 |
| output | 10 | 5 | 10 | 1 |

We are dealing with a simple economy that produces iron,coal,corn and bread. This is a standard format, albeit tiny, input output table. Read down the iron column. Right down at the bottom you see an output row. This says that at the start the economy produces 10 units of iron, and to make that iron it uses 0.1 unit of existing iron, and 2 units of coal plus 0.3 units of labour. Dont worry what the units are for the moment. In principle they can be anything, tons, kgs cubic feet etc, provided that each type of product is consistently measured in its own unit. So you could measure iron in million kg, coal in million lbs, bread in thousands of loaves etc, so long as each mention of iron in the tables is always in million kg, each mention of coal is in million lbs etc. Similarly to produce 5 units of coal, the economy uses up 1 unit of labour and 1 unit of coal. For each industry you have a column describing how it is made, and each row represents intermediate uses of a given product.

All of these are flows of intermediate products. That is to say flows that are immediately used up as raw materials. The flows do not include flows to replace capital stock used up. For that we need two more tables. The next is the capital stock table, `cap.csv`,

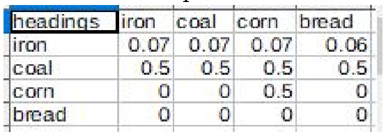


| headings | iron | coal | corn | bread |
|---|---|---|---|---|
| iron | 10 | 7 | 1.4 | 1 |
| coal | 2 | 1 | 0.04 | 0.1 |
| corn | 0 | 0 | 1 | 0 |
| bread | 0 | 0 | 0 | 0 |

This says that to produce the current output of iron, we need a capital stock of 10 units of iron ( think big machines ), and a buffer stock of 2 units of coal – think piles of it in the ironworks. These are not flows, these are the stocks needed

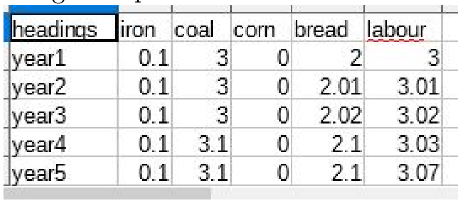
[1]both the lp-solve package and the more advanced harmony package are available at https://github.com/wc22m/5yearplan

to continue production. Note that there is no output or labour row for this table, since the labour and the output are assumed to be the same as in the `flows.csv` table. However capital stocks wear out. And some stocks wear out more quickly than others. A van, for example, wears out faster than a railway locomotive. So a long term plan needs to know how fast each type of product wears out. This is specified in the depreciation table `dep.csv`, which in our example contains:



| headings | iron | coal | corn | bread |
|---|---|---|---|---|
| iron | 0.07 | 0.07 | 0.07 | 0.06 |
| coal | 0.5 | 0.5 | 0.5 | 0.5 |
| corn | 0 | 0 | 0.5 | 0 |
| bread | 0 | 0 | 0 | 0 |

This table tells you the rate at which the stock of each type of means of production depreciates each year. So 0.07 or 7% of the iron used to make iron wears out each year, whereas the buffer stock of seed corn to make corn, deteriorates much faster, half of it is lost per year. Finally we have a table labtarg.csv, that specifies the target outputs and available labour force each year of the plan.



| headings | iron | coal | corn | bread | labour |
|---|---|---|---|---|---|
| year1 | 0.1 | 3 | 0 | 2 | 3 |
| year2 | 0.1 | 3 | 0 | 2.01 | 3.01 |
| year3 | 0.1 | 3 | 0 | 2.02 | 3.02 |
| year4 | 0.1 | 3.1 | 0 | 2.1 | 3.03 |
| year5 | 0.1 | 3.1 | 0 | 2.1 | 3.07 |

These specify the target levels of final consumption of each product each year. It is important to note that this is not the total output, since that would also include outputs of raw materials and replacement capital goods. Instead we specify the plan in terms of the output available for consumption. So in the first year we want 0.1 unit of iron for direct consumption – knives and forks, pots and pans, 3 units of coal for heating and cooking, and 2 units of bread. The labour force available to the economy will be 3. The following year the labour force has grown by 0.01 to 3.01 and, with this larger labour force we aim to produce slightly more bread. And so on for each year. You can add or remove years from the plan by editing the table.

The planning algorithm seeks to maximize the fulfillment of these plan targets for each year of the n year plan. I does this by generating a programme for the lp-solve language which is printed on the standard output stream.

It is made up of a long series of inequalities preceded by a maximisation objective. In this case it starts out

```
max:targetFulfillmentForYear1 +targetFulfillmentForYear2
  +targetFulfillmentForYear3 +targetFulfillmentForYear4
  +targetFulfillmentForYear5;
.....
```

It is attemptint to maximise the sum of the individual year plan fulfillments. For each year the plan fulfillment is specified in terms of meeting a specified ratio of outputs. Since the target was to produce at least 0.1 of a unit of iron, exact fullfilment of the plan can not be greater than 10 times the final consumption of iron in year 1, with similar rules for coal and bread.

```
targetFulfillmentForYear1 <=10.0  finalConsumptionOfiron1;
targetFulfillmentForYear1 <=0.333 finalConsumptionOfcoal1;
targetFulfillmentForYear1 <=0.5   finalConsumptionOfbread1;
```

Initial resources are specified as other constraints. So the labour available has to be $\geq$ the labour used in each industry, and $\leq$ than the pregiven supply for the year. Similar constraints apply to the pre allocated stocks of capital goods. It is assumed that these are fixed capital, ie, they can not be transfered between industries.

```
labourForYear1>= labourForiron1 +labourForcoal1
                      +labourForcorn1 +labourForbread1;
labourForYear1<= 3.0;
outputOfiron1<= 1.0 capitalstockForironMadeUpOfiron1;
outputOfiron1<= 100.0 flowForironOfiron1;
```

Additional rules stipulate what is to be done with the output:

```
accumulationOfcoal2>=accumulationForironOfcoal2
 +accumulationForcoalOfcoal2
 +accumulationForcornOfcoal2 +accumulationForbreadOfcoal2;
productiveConsumptionOfcoal2>= flowForironOfcoal2
 +flowForcoalOfcoal2
 +flowForcornOfcoal2 +flowForbreadOfcoal2;
finalConsumptionOfcoal2<=outputOfcoal2 - accumulationOfcoal2
   -productiveConsumptionOfcoal2;
```

The rules for depreciation and capital accumulation serve to tie together the plans for different years thus

```
depreciationIncoalProductionOfcoal2 =0.5 capitalstockForcoalMadeUpOfcoal2;
capitalstockForcoalMadeUpOfcoal2<=capitalstockForcoalMadeUpOfcoal1
 + accumulationForcoalOfcoal1
 - depreciationIncoalProductionOfcoal1;
```

Assume that the linear optimisation program has been stored in the file `plan.lp` we can run the program thus

```
lp_solve <plan.lp|sort >plan.txt
```

The final solution to the plan, is given in a fairly verbose and self descriptive format, which specifies the output for each product each year, the amount of investment of each type in each industry each year, the labour employed etc.

## 3. Complexity of lp-solve

The complexity of a planning problem will clearly depend on how many products are being tracked and on the time period over which the plan is to run. To evaluate this tests were performed using input output models of increasing size and varying number of years of plan horizon.

The models were synthetically generated using random number generators and were structured to ensure that they were all economically feasible - that is to say the IO tables all produce a material surplus product. It is known that IO tables grow in sparseness as they become increasingly disagreggated[8]. Since the `nyearplan` software takes advantage of sparsity, only generating constraints associated with non-zero matrix cells, it is important that the test models show the $N \log N$ growth in the number of non-zero cells that has been observed in real tables. The random IO table generator is thus set up to produce tables with this statistical property.

Measurements show that the `lp-solve` system has a complexity order of $N^3$ where $N$ is the number of industries. With respect to plan horizons, a 2 year with $N$ industries takes about 6.5 as long to evaluate as a one year plan, a 5 year plan takes about 72 times as long. The overall complexity seems to be of the order of

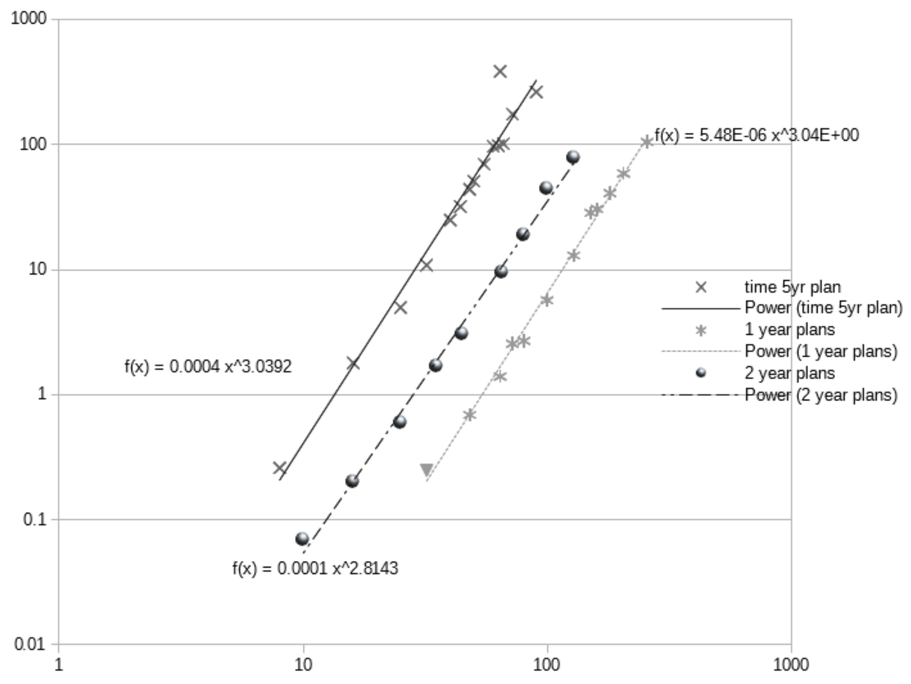


FIGURE 3.1. The run time of lp-solve for economic planning appears to be of order $N^3$. Performance was measured on and small Odroid micro computer running Ubuntu.

TABLE 1. Projected time to compute large plans with lp-solve

| industries | 1 year plan | 5 year plan |
|---|---|---|
| 500 | 0.24hr | 17hr |
| 5000 | 11 days | 2.2 years |
| 50000 | 33years | 2417years |

$N^3Y^{2.6}$ with $Y$ the number of years. This is polynomial but of a sufficiently high order as to make large models as shown in Table 1.

Whilst the linear solver used in lp-solve is practical for macro economic planning, it would be impractical for detail planning. The tests were done on a single 64 bit ARM core running at 1.6Ghz. Parallelism of the sort available in modern super computers would allow the times to be massively reduced, but it is far preferable to start off with a less complex algorithm. The relatively poor performance of standard linear optimisers on disaggregated plans was known to Soviet economists.This result presumably lay behind the data cited by the late Prof Nove in his book[9] where he gives astronomically long times for computers to construct optimal plans for whole economies. But that does not exclude the possibility that there may be iterative algorithms of low complexity that can achieve a good result on the same problem.

There is a long history of debates by economists on the feasibility of planned economic calculation[10, 11, 12, 13, 14, 15, 16, 17]with one school asserting it was feasible, another saying it was impossible. But given that any explicit calculation or algorithm that humans can do, can also, in principle be done by computers.

Given further, that the complexity order of an algorithm does not change depending on whether people do it by hand or a computer does it. It follows then that the existence of functioning market economies is proof that low complexity coordination algorithms exist. It is clearly not the case that market economies depend on an algorithm of order $N^3$ or they would never have grown to be able to produce the hundreds of millions of products[18] that actually are on sale. The Harmony Algorithm[6]draws on ideas from marginalist economics and from neural nets[19] to derive an iterative planning technique. This has previously been shown to have $N \log N$ complexity for single year plans. Here we present an implementation suitable for multi-year plans.

## 4. The Harmony Algorithm

The algorithm takes the plan problem as defined by Kantorovich. In the current implementation the interface is identical to that used in section 2.

A technique is defined as producing specified amounts of one or more outputs, using specified amounts of one or more inputs, which we can represent as:

$$P : a_1x_1, a_2x_2, a_3x_{3,}, ... \rightarrow p_1y_1, p_2y_2, P_3y_3, ... \tag{4.1}$$

so that technique $P$ uses $a\_1$ units of input $x_1$, $a_2$ of $x_2$ etc to produce $p_1$ of output $y_1$ etc.

Clearly, if we only use one output, any Leontief form IO table can easily have its columns represented as such techniques. In the algorithm products and inputs $y_i, x_j$ are assigned index numbers for identification. A given product may appear both as an output of one technique and the input to other techniques, it may be pre-given resource or an item of final consumption that never appears as an input.

Time periods can be linked by investments or by the persistence of capital goods between periods. One way to represent capital goods persistence, described by Sraffa[20], is to describe every production process as outputing partially used capital goods as joint products. Thus a technique operating in time period $t_0$ might produce a main product to be consumed in this period along with a set of partially used capital goods to be available in period $t_1$. Alternatively one can model investment in time $t_0$ as producing capital goods available for use in periods $t_{1..h}$where $h$ is the depreciation horizon. In either case joint production techniques and as a result for years $t_{2..}$ we have multiple alternative techniques which can supply the capital goods: investments in $t_0, t_1$. The combination of joint production and multiple techniques was not present in the original published algorithm. This is relevant, because questions have been raised[21]as to whether the Harmony Algorithm is applicable in these cases.

The key feature of the algorithm is that instead of specifying a hard rule that plan outputs should be supplied in fixed proportions a function, called the Harmony function, is used to weight deviations from fixed proportionality of output.

The Harmony function is supposed to mimic the principle of positive but diminishing marginal utility. What is required is a function whose value rises as plan fullfillment approaches but which rewards overfullfilment less than it punishes underfulfillment. As implemented the function is

$$h(t,n) = \begin{cases} S - \frac{S^2}{2} & S < 0 \\ \ln(S+1) & S \geq 0 \end{cases} \tag{4.2}$$

where

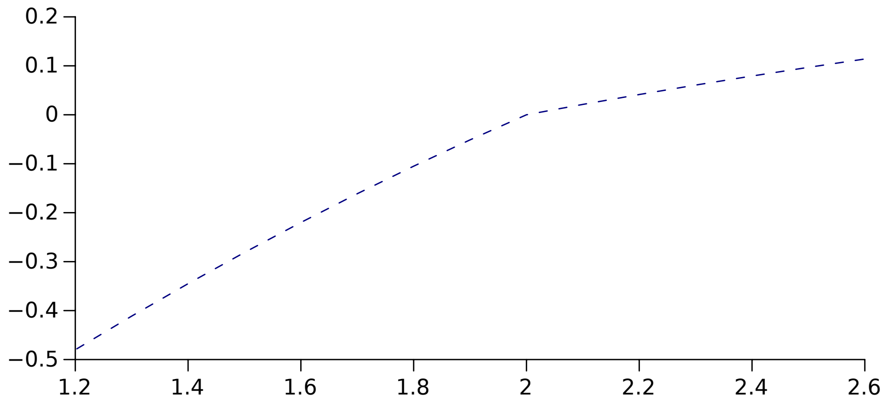


FIGURE 4.1. Harmony function for a planned output of 2 units. Note Harmony =0 when plan is on target.

$$S = \frac{n-t}{t} \tag{4.3}$$

and $t$ is the plan target, and $n$ the net output of the corresponding product. The shape of the function is shown in Figure 4.1. It is continuous, has a positive first and a negative second derivative. Given a technology complex $C$, a plan ray $P$, a vector of initial resources $R$ and an initial intensity vector $I$ specifying the scale of production of each of the techniques it is simple to compute the net output vector $N$ and from this we derive the Harmony vector $H$ and the harmony gradient vector $\nabla H$. For non-final goods the harmony derivative is obtained from the mean of the harmony derivatives of the goods they contribute to multiplied by their marginal physical product in each context.

For each production technique $p$ we derive a harmony gain rate per production cycle $G_p$ which is given by

$$G_p = \frac{\sum p_i dh/dy_i - \sum a_j dh/dx_j}{\sum a_j dh/dx_j} \tag{4.4}$$

where the $p_i, a_i$ are the production and use coefficients in eqn. 4.1. This harmony gain rate per cycle is a computational analogue of the rate of return on capital.

The algorithm iterates around three steps:

(1) A Newton Raphson phase which estimates the mean harmony $\mu_H$and attempts to move the scale of production of all industries towards the level at which they would be operating at mean harmony. Industries $i$ with $h_i < \mu_H$ are expanded the others are shrunk. Intercepts with the mean are determined using the gradients $\nabla H$ and a scaling factor $\psi < 1$ is used to move the industries part way towards the mean. The adjustment of any one industry will have second order effects in terms of the plan fullfilment of raw materials and intermediate products it uses up.
(2) A hill creep phase analogous to the movement of capital in a market economy towards those branches where the rate of return is highest. The intensity $I$ of each technique $j$ is adjusted such that $I_j = I_j \times (1 + \sigma(G_j)\phi\psi)$ where $\phi <$ 1is another rate control constant and $\sigma(x)$ is a sigmoid function

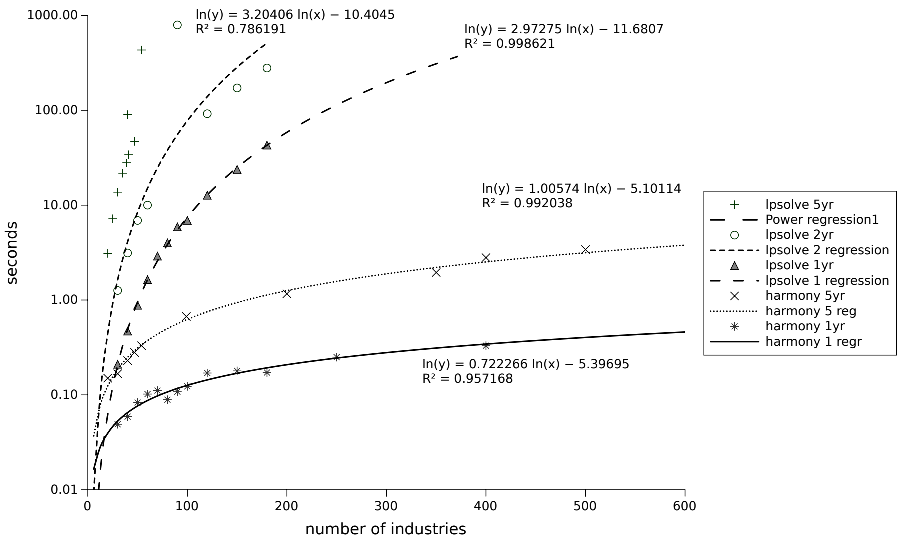


Figure 4.2. A comparison of the performance of the lp-solve and Harmony based plan solvers. Note that the Y axis is a log scale. Tests run on an AMD A12 with a 1Ghz core clock using Ubuntu under VirtualBox with 2 cores available.

to map $-\infty..\infty$ into the range $-1..1$. A suitable sigmoid function is

$$\sigma(x) = \begin{cases} \frac{x}{1+x} & x > 0 \\ 0 & x = 0 \\ -\sigma(-x) & x < 0 \end{cases} \tag{4.5}$$

(3) A final consistency adjustment phase to make sure that there is not a negative net product of any type. For this the vector of net products is searched for negative values. If any are found then the relative scale of use reduction that would be required to remove the deficit is calculated. All techniques which use the product are then scheduled be reduced by at least this amount. If a technique is scheduled to be reduced by 5% due to a shortage of coal and 4% due to a shortage of iron, then the total reduction will be 5% not 9%.

**Intialisation.** The intensity of all processes is initialised to 20% and initial capital stocks for each non zero capital matrix for each year are derived from the approariately depreciated initial stocks. For each year other than the last, an accumulation technique is defined which consumes one unit of the appropriate capital type in year $t$ and delivers one unit of the appropriate type of capital, appropriately depreciated, each subsequent year. As with the linear programming solver described in section 2, it is assumed that the capital stocks are fixed and not redeployable outside of the sector in which they were originally invested.

**Results.** For both the lp-solve and the Harmony based planning algorithm timings exclude the initial reading in and parsing of the IO tables. The Harmony solver is so fast that computation time is completely dwarfed by the IO time otherwise on large examples. Results are shown in Figure 4.2. It is clear that over the range of plans tested, the Harmony solver is very much faster than the lp-solve version. From the graphs shown, it appears that the Harmony solver is, for a given number of years, approximately linear in the number of industries being planned. More plausibly, it is of order $N \log N$, but this is hard to distinguish given the range of the dataset.

The input data format, using IO matrices, is very inefficient for large numbers of industries. The space taken up by the planning tables and by the printed results grows as order $N^2$. At present this limits the number of industries worked with to under 1000 for practical purposes. Above that level one runs out of heap space in the java module that parses spreadsheets.

From the standpoint of the underlying planning algorithm that is not a problem. Given data provided in a more compact form, for example from relational databases, space would not be a problem until much larger numbers of products or industries are encountered, since the storage used for the actual calculation grows only in proportion to the non-null entries in the production matrix. This indicates that were only two cores of the same power as used earlier available, a 50,000 industry 5 year plan could be optimised in under ten minutes, which compares favourably with Table 1.

Use on larger scales would be helped by massive parallelism, but the algorithm is well suited to this. Each of the steps of the main algorithm may be run in data parallel mode provided that all cores synchronise access to the shared vectors $I, H, \nabla H$ between steps.

Overall we conclude that given the resources available to the Chinese computing industries, the objectives set by Mr Ma, are indeed technically feasible.

## References

[1] L.V. Kantorovich, *The Best Use of Economic Resources*, Harvard University Press, 1965.

[2] L.V. Kantorovich, "Mathematical Methods of Organizing and Planning Production", *Management Science*, 6(4), 1960, pp. 366–422.

[3] GB Dantzig and P. Wolfe, "The decomposition algorithm for linear programming", *Econometrica*, 29(4), 1961, pp. 767–778.

[4] G. Dantzig, " Linear Programming", *Operations Research*, 50(1), 2002, pp. 42–47.

[5] M Berkelaar, et al., "lp solve:(Mixed Integer) Linear Programming Problem Solver, 2004", *URL: ftp://ftp es ele tue nl/pub/lp solve.*

[6] W. P. Cockshott, "Application of artificial intelligence techniques to economic planning", *Future Computing Systems*, 2, 1990, pp. 429–443.

[7] John Thornhill, "The Big Data revolution can revive the planned economy", *Financial Times*, 2017-9-4.

[8] Michael Reifferscheidt and Paul Cockshott, "Average and marginal labour values are $\mathbf{O}\ n \log(n)$ — a reply to Hagendorf", *World Review of Political Economy*, 5(2), 2014, pp. 258–275.

[9] Alex Nove, *The Economics of Feasible Socialism*, George Allen and Unwin, London, 1983.

[10] L. von Mises, "Economic calculation in the socialist commonwealth", in *Collectivist Economic Planning*, edited by F A Hayek, Routledge and Kegan Paul, London, 1935.

[11] L. von Mises, *Socialism: An Economic and Sociological Analysis*, Johnathan Cape, 1951.

[12] D. Greenwood, "Commensurability and beyond: from Mises and Neurath to the future of the socialist calculation debate", *Economy and Society*, 35(1), 2006, pp. 65–90.

[13] F. A. Hayek, "The use of knowledge in society", *American Economic Review*, 1945, pp. 519–530.

[14] W. Marciszewski, “Hypercomputational vs. Computational Complexity A Challenge for Methodology of the Social Sciences”, *Free Market and Computational Complexity Essays in Commemoration of Friedrich Hayek (1899-1992) Series: Studies in Logic, Grammar and Rhetoric*, 5, 2002, p. 18.

[15] Paul Cockshott and Allin Cottrell, “Information and Economics : a critique of Hayek”, *Research in Political Economy*, 18(1), 1997, pp. 177–202.

[16] Oscar Lange, *On the Economic Theory of Socialism*, University of Minnesota Press, 1938.

[17] Oscar Lange, “The computer and the market.”, in *Socialism, capitalism and economic growth: essays presented to Maurice Dobb.*, Cambridge University Press., 1967.

[18] Paul Grey, “How Many Products Does Amazon Sell?”, , 2015, URL `https://export-x.com/2015/12/11/how-many-products-does-amazon-sell-2015/`.

[19] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, 1986, pp. 194–281.

[20] Piero Sraffa, *Production of commodities by means of commodities*, Cambridge University Press, Cambridge, 1960.

[21] Cosma Shalizi, “The impossible takes a little longer”, , 2012, URL `http://bactra.org/weblog/919.html`.