Choosing Algorithm Parameters Strategically

Erlendur Smári Þorsteinsson
esth@cmu.edu

Supervised by: Prof. Egon Balas

A first summer paper submitted to the

Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213

1st reader: Prof. Egon Balas, 2nd reader: Prof. Gerard Cornuéjols

December 12th, 1997

Abstract

In this paper we examine the potential use of artificial intelligence in choosing values for free parameters in the software implementation of algorithms. As a particular example we examine a branch-and-cut procedure for solving integer programs and an implementation of that algorithm called MIPO. We examine a particular parameter in that procedure called the skip factor whose value decides how often cuts are added to the problem description. We demonstrate how carefully choosing the value for this parameter using neural networks results in a decrease in solution time compared to using a fixed value or a value suggested by MIPO.

1 Introduction

When a user is faced with a new complicated algorithm or program a common approach is often to accept the default values suggested for choices or free parameters. This often leads to non-optimal performance since these values can greatly affect the performance of the algorithm.

When deciding on the default values for the free parameters the developer has often tuned the parameters through a feedback loop that included the developer as a learning component [4]. The quality of the results depends on the problem pool available to the developer and is not automatically transferred to new instances.

Also, this learning process is often poorly documented, hindering reproduction of results.

In this paper we are going to examine how it is possible to use artificial intelligence to tune the parameters automatically so the developer does not become a key element in the optimal performance of the algorithm. To this extent we are going to examine a key decision in a branch-and-cut integer program solving, the question whether you should branch or cut at a given node in the search tree. We are going to use a neural network to estimate at what interval in the node sequence we should cut.

We note that there is an inherit trade-off in the general process. On one hand the algorithm will become more complex and more time will have to be spent on developing it. The developer has to describe his expertise in more detail, in particular the usage of the algorithm. On the other hand the algorithm becomes more self-contained and autonomous, simplifying its usage. Thus only the end user will reap the benefits and the developer is faced with a more arduous task than before.

1.1 Static versus Dynamic Tuning

There are two types of parameters an algorithm can have, parameters that do not change during each run (static) and parameters that possibly change during each run (dynamic), each dividing into further subcategories.

Static parameters can be the results of extensive testing or simply design decisions and fixed forever without regard to the problem instance at hand. They can also take into account the problem instance to be solved. By examining it in some detail it may be possible to determine a more appropriate parameter settings than one-size-fits-all. After the solving process starts, however, the parameters do not change.

Dynamic parameters have the ability to change during the solution process based on local properties of the particular instance and the solution process history. If required they are changed according to rules saying when to update the values and how to update them. We note that static parameters can be viewed as dynamic parameters where the update rule is to never update the values.

1.2 Organisation of the paper

This paper and the underlying research work is to some extent based on a paper by Balas, Ceria and Cornuéjols [2] and MIPO, an integer solver developed by the same parties.

Section 2 discusses the branch-and-cut procedure. Section 3 deals with artificial intelligence and neural networks. Finally, in section 4 we describe the methodology used in our experiments and compare this procedure to previous procedures and naive decisions.

2 Branching versus Cutting

We assume that the reader is familiar with the common approach of branch-and-bound to solve integer programs. Although this method can be quite effective in solving small and medium scale problems, it can be difficult to solve large scale integer programs using branch-and-bound. To be able to solve large problems it is often necessary to strengthen the formulation using automatic problem pre-processing or cut generation.

Cutting-planes is another common method to solve or facilitate the solution of integer programs. It is often a part of the pre-processing but it can also be used as a solution method where solutions to the linear program relaxation that are not in the integer polyhedron are cut off. By adding the cuts to the formulation we aim at getting a description of the convex hull of the integer polyhedron in the vicinity of the solution.

The problem with this method has been that a special purpose algorithm with a special purpose cut generator has been required for each class of problems to fully exploit its combinatorial structure [2]. Another approach, originally by Gomory [10], is to use general purpose cuts. Recently, lift-and-project cuts have been shown to be effective general purpose cuts for mixed 0-1 programs [1], [2].

Padberg and Rinaldi recently proposed a different approach to solving integer programs by intertwining branch-and-bound and cut generation [15]. Instead of separating the cutting-plane phase and the enumeration phase, cuts are now added in the enumeration phase as needed. Their implementation was to add cuts at every node in the search tree until the benefits from adding the cuts were below a certain threshold and then resort to pure branch-and-bound. Balas et al. [2] improved upon this procedure by generating cuts at regular intervals in the search tree, stipulated by the user. Another improvement was using general purpose cuts, such as lift-and-project cuts or Gomory cuts, within a branch-and-bound framework resulting in a versatile algorithm capable of handling many classes of problems.

A crucial idea within this framework is cut lifting. The cuts generated at each node in the search tree are only valid at that node and its descendants. To make them valid throughout the search tree they have to be lifted, i.e. we have to find the tightest cut possible which is valid throughout the search tree, such that its projection onto the variable space at the current node is the original cut.

Balas et al. also came up with a formula to estimate a reasonable skip factor. The answer to the question "To cut or not to cut" is however much more complicated than that. Although it may be reasonable to use the same skip factor for all problems, for all classes of problems, or for each individual problem, it can be beneficial to vary the skip factor within the solution process for each problem.

One idea might be to use a fixed rule and let the skip factor gradually become larger, a similar idea to simulated annealing. We are not going to explore this avenue but rather focus on how artificial intelligence can help us in estimating the correct skip factor.

3 Neural Nets

For background it is necessary to include in this paper a short discussion on artificial intelligence.

But what is artificial intelligence? As the human mind is difficult to understand the definitions vary. Most definitions can though be classified into one of the following four categories:

Systems that think like humans.	Systems that think rationally.
Systems that act like humans.	Systems that act rationally.

Referring to the categories on the right, artificial intelligence can be thought of as the branch of computer science that is concerned with the automation of intelligent behaviour and thinking [16]. We are only going to be focusing on a small subset of this, learning functions and classifying patterns. A simple idea to accomplish this are neural networks, an idea that would fall into the first category above.

3.1 Neural Networks

The idea of neural networks is based on imitating the way the brain works. The fundamental unit of the brain is the neurone. Each neurone is composed of the cell body, *soma,* a number of fibres called *dendrites* and a single long fibre called the *axon*. The axon branches out and connects to the dendrites and soma of other neurones. The connecting junction is called a *synapse*.

Neurones communicate by electrochemical reaction. Chemicals are released from the synapses, enter the dendrite, raising or lowering the electric potential of the soma. When the potential reaches a threshold an electrical pulse is sent down the axon eventually reaching the synapses and releasing transmitter chemicals into other cells.

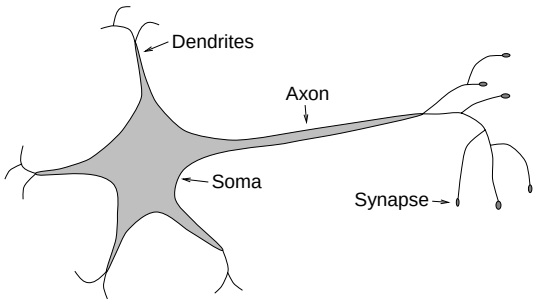

Figure 1: Nerve cell.

Suppose we are interested in approximating a function $k : P \to \mathcal{N}$, where in this paper the function k is the skip factor and P is the domain of integer programs. We let r be a vector of parameters and approximate $k(p)$ by $\tilde{k}(p, r)$, $p \in P$, where $\tilde{k}$ is

a function easy to evaluate once r has been fixed. We then try to choose r so as to minimise the distance between k and $\tilde{k}$.

The function $\tilde{k}$ will be represented by a neural network. A simple form is a linear approximation

$$\tilde{k}(p,r) = \sum_{j=1}^{R} r(j)\phi_j(p),$$

where $r(j)$, $j = 1, \ldots, R$, are the components of the parameter vector r (weights) and ϕ_j are fixed functions (inputs).

The input functions ϕ_j are feature extractions, extracting relevant aspects of the problems for input into the network.

Training the neural net involves minimising the error, e.g. by minimising

$$\sum_{p} \Big(k(p) - \tilde{k}(p,r)\Big)^2,$$

for known data pairs $(p, k(p))$. In the linear case this reduces to

$$\sum_{p} \Big(k(p) - \sum_{j} r(j)\phi_j(p)\Big)^2.$$

A common nonlinear form is neural network with a single hidden layer. This involves the use of a sigmoid function, or threshold function, for which a common choice is

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}.$$

The formula is

$$\tilde{k}(p,r) = \sum_{j} r(j)\sigma\Big(\sum_{i} r(j,i)\phi_i(p)\Big),$$

and is derived from the network in figure 2.

The weights now consist of $r(j)$ and $r(j,i)$. The values of $\phi_i(p)$ are the inputs to this network. Their values are then combined for input into the sigmoid functions, weighted by $r(j,i)$. The sigmoid functions signal if the combined input exceeds the threshold. The output is then the combination of the sigmoid functions, weighted by $r(j)$.

This can be made more complicated by adding layers, interleaving linear and sigmoidal layers, and also by having more outputs. While it is often relatively easy to decide on the inputs and outputs, the tricky part is deciding on the number of hidden layers and number of units in each hidden layer.

As before, training the neural net involves minimising some measure between $k(p)$ and $\tilde{k}(p,r)$, e.g. the Euclidean distance, for known datapairs. With a nonlinear architecture the least squares problem of minimising the Euclidean distance cannot be reduced to a linear algebraic problem and must be solved using nonlinear programming methods, many of which are discussed in [9] and [12].

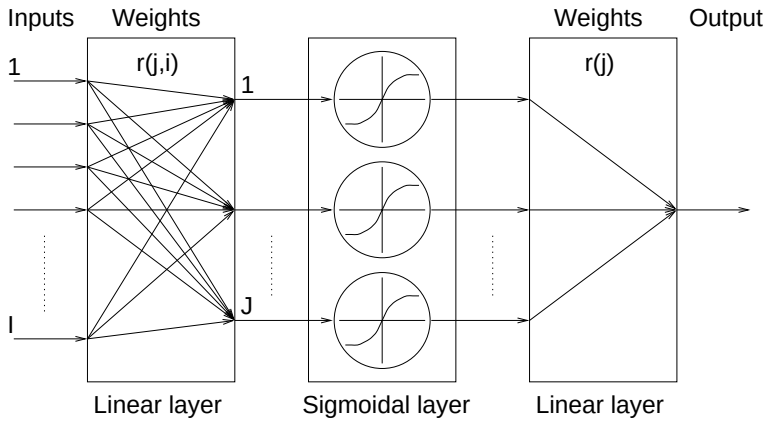

Figure 2: A neural network with a single hidden layer.

3.2 Leave-One-Out Cross-Validation

To test the prediction accuracy of the neural network, one would normally train the network on a training dataset and test on a separate testing dataset, each set of reasonable size. By doing this repeatedly for different training and testing datasets the accuracy of the network could be validated. Due to limited data available another method had to be used to validate the network.

Leave-one-out cross-validation is a way to check how well a model will generalise to new data. For each point in the dataset we compare the output from the network when predicting the value for that point, when it is included in the training set and when is omitted from the training set. The error is the difference in the outputs.

To calculate the error made by the network, we used a variant of this technique. We split the data up in two sets, a test set consisting of only one data point and training set consisting of all the other points. We trained and tested on these sets, recorded the result, and repeated the procedure for all the points, each time putting a different point into the test set. The error made by the network was the squared difference between the value so predicted by the network and the true value.

4 Computational Results

In this section we focus on the methodology used in performing the experiments and report on the results we got.

We used MIPO v0.91, which implements lift-and-project cuts in a branch-and-cut framework, and CPLEX 2.1m on an HP A 9000/720 workstation with 64Mb of memory.

Problem name	Constraints	0-1 variables	Continuous variables	Problem density
bm21	20	23	0	88.48%
bm23	20	27	0	88.52%
egout	129	55	86	1.72%
fix3	478	378	500	0.42%
fxch3	161	141	141	1.23%
gen	780	144	720	0.38%
khb05250	152	24	1326	1.34%
lp4l	85	1086	0	5.07%
lseu	28	89	0	12.40%
misc01	54	82	1	16.62%
misc03	96	159	1	13.37%
misc05	300	74	62	7.22%
mod008	6	319	0	64.94%
mod013	62	48	48	3.23%
p0033	15	33	0	19.80%
p0201	133	201	0	7.19%
pipex	25	48	0	16.00%
rgn	104	100	80	2.88%
sentoy	30	60	0	100.00%
stein27	118	27	0	11.86%
truck4	15	152	0	13.33%
tsp43	143	1117	0	5.26%
utrans.1	103	80	86	1.94%
vasilis2	305	86	264	1.16%

Table 1: Problem characteristics, first batch.

4.1 Basic testbed

In order to test the advantage of using neural networks to predict the skip factor, we solved a number of problems using different values for the skip factor. The results were used to train the network and test its abilities. The characteristics of these problems can be found in table 1. Descriptions of the problems and known references can be found in appendix A.1.

4.2 Extended testbed

To test the network further we solved some additional problems using only a limited number of different values for the skip factor. The results were only used to test the abilities of the network, not to train it. The characteristics of these problems can be found in table 2. Descriptions of the problems and known references can be found in appendices A.2 and A.3.

Problem name	Constraints	0-1 variables	Continuous variables	Problem density
mod010	146	2655	0	2.89%
cracpb1	144	518	55	5.04%
vasilis_2	305	86	264	1.16%
utrans.2	480	120	120	1.33%
utrans.3	182	142	142	1.10%
c-fat200-1	1919	200	0	8.74%
vasilis	176	66	119	2.25%
vasilis_3	305	86	264	1.16%
martin	620	2380	0	0.34%
genova6xs	98	904	0	5.02%
vpm1	486	168	210	0.54%
san200_0.9_3	1126	200	0	1.37%
l152lav	97	1989	0	5.14%
misc07	211	259	0	15.35%
stein45	331	45	0	6.94%
vasilis_1	130	55	11	4.84%

Table 2: Problem characteristics, second batch, small and large problems.

The distinction between the first batch and the second batch, and between small and large problems will be discussed in greater detail later in this section.

4.3 Other variables

There are other variables than the skip factor that can effect the solution time. To minimise their effect on these experiments those variables were given fixed values, mostly based on values found in [2] and also through private communication [8]. The values for some important variables are noted below.

In our experiments, cuts are generated in one round per node for all the 0-1 variables which are fractional in the current solution or at most 40. In [2] the upper limit was 40 for problems with up to 400 0-1 variables and 80 for problems with more 0-1 variables. Most of the problems we use have less than 400 0-1 variables and so it was decided to use only one upper limit regardless of the number of 0-1 variables.

As mentioned in [1] there are a number of ways to normalise the lift-and-project cut added. In our experiments we only use normalisation 3, compared to normalisation 1, 2 and 3 in [2].

When selecting the next node to process we used the best-bound rule, i.e. selecting the node with the largest objective value. When selecting a variable to branch on we used the Hoffman-Padberg strategy, described in [2].

We also used cut strengthening and cut lifting.

4.4 Automatic skip factor

MIPO offers two choices when selecting the skip factor, either it is fixed by the user to a certain value or MIPO selects a value. The *automatic* value is determined by a built-in formula [2],

$$k = \min\left\{32, \left\lceil \frac{f}{cd \log_{10} p} \right\rceil \right\},$$

where f is number of cuts generated at the root node, d is the average distance cut off by the cuts added at the root, p if the number of 0-1 variables and c is a constant. In our tests we set $c = 10$.

In the current implementation the value is fixed during the solution process for each problem (static tuning). We will discuss later some modifications to MIPO to dynamically tune the skip factor during the solution process.

4.5 First experiments

The aim of the first experiments was to get data to train the neural network. We had decided, based on recommendations in [2], to use skip factor between 1 and 32. The reason for this is to ensure the robustness of algorithm. Although the automatic method or the neural network (the *prediction*) might recommend cutting more seldom it is advisable to keep an upper bound on how seldom the algorithm cuts.

We had therefore to find out what value for the skip factor resulted in the least solution time. To this extent we selected a number of problems, the first batch, such that they could be solved 32 times in reasonable time, using the values 1 to 32 for the skip factor. While solving those problems we recorded a number of parameters that would possibly be used as inputs into the neural network later on.

One of the items we noticed during those experiments was that the curve of CPU time as a function of the skip factor can often be far from smooth. While some problems had a clearly defined minimum, the curves for other problems could be quite erratic, sometimes even as if certain values for the skip factor resulted in less CPU time just by chance.

After running the experiments and determining the value for the skip factor resulting in the least solution time, we trained the network in recognising the true value, using the leave-one-out technique outlined in section 3.2. Since the skip factor should be an integer between 1 and 32, the output from network was rounded to the nearest integer in that range.

Based on previous experience and partly because of limited data, we decided on using a single layer neural network with three inputs. The first input is the automatic value suggested by MIPO. The second is the cut-off distance at the root, i.e. the average distance cut off by the cuts at the root. Both these inputs measure in some way the quality of the cuts, but do not include the time spent on generating the cuts compared to the advantage of using them. Thus we decided the third input

Problem name	Auto. value	Pred. value	True value	Time auto.	Time pred.	Time true	Time 8	Time 10
bm21	5	12	29	12.26	8.10	5.92	8.74	7.77
bm23	5	15	26	13.44	7.41	6.11	10.42	9.91
egout	1	21	9	23.02	31.38	21.91	25.35	25.12
fix3	2	14	1	225.72	161.75	134.65	167.55	154.84
fxch3	2	7	19	328.72	268.23	181.78	298.66	249.35
gen	2	9	3	241.52	217.18	177.39	204.29	278.09
khb05250	1	10	18	195.25	58.11	55.10	74.49	58.11
lp4l	32	30	1	95.01	87.06	59.34	121.37	100.12
lseu	9	9	20	47.32	47.32	30.03	81.23	41.19
misc01	8	8	32	57.86	57.86	26.10	57.86	47.56
misc03	30	28	28	67.55	63.21	63.21	162.43	126.08
misc05	13	13	20	239.33	239.33	167.32	277.57	233.89
mod008	21	20	13	178.36	206.85	124.32	161.23	137.46
mod013	1	8	18	59.45	18.28	11.37	18.28	16.28
p0033	1	26	20	9.97	4.92	2.84	4.57	5.54
p0201	9	9	18	452.13	452.13	250.13	422.30	474.73
pipex	7	9	29	15.73	10.97	9.90	19.97	12.41
rgn	5	7	8	69.81	76.61	50.98	50.98	74.97
sentoy	11	11	16	11.38	11.16	10.05	14.60	12.73
stein27	9	10	30	262.08	258.85	204.45	278.06	258.85
truck4	3	5	13	25.61	11.46	5.14	6.51	11.14
tsp43	8	8	10	273.54	273.54	272.87	273.54	272.87
utrans.1	3	8	14	220.71	202.68	107.27	202.68	151.00
vasilis2	32	32	32	51.80	51.80	51.80	81.62	74.57
			Sum	3177.6	2826.2	2030.0	3024.3	2834.6
				156.5%	139.2%	100.00%	149.0%	139.6%

Table 3: Results for the first batch, skip factor and CPU seconds.

would be

$$\frac{l-b}{lt},$$

where l is the solution to the linear program relaxation, b is the lower bound after adding the cuts and t is time spent adding the cuts at the root. This input measures directly the normalised gain per time unit of using the cuts.

The results are summarised in table 3. We notice that overall we have improved upon the automatic choice using the neural network.

It is also interesting to compare the results with the time spent if the values 8 and 10 are used for the skip factor. Eight is the value previously found to be the best overall value, i.e. if the same value is used for all problems. Ten is a value a novice user might decide to use. The only reason for that is that it is a nice, round number between 1 and 32. Life is full of such decisions, such as 5% confidence intervals,

Problem name	Auto. value	Pred. value	True value	Auto. vs. true	Pred. vs. true	8 vs. true	10 vs. true
bm21	5	12	29	207.1%	136.8%	147.6%	131.3%
bm23	5	15	26	220.0%	121.3%	170.5%	162.2%
egout	1	21	9	105.1%	143.2%	115.7%	114.7%
fix3	2	14	1	167.6%	120.1%	124.4%	115.0%
fxch3	2	7	19	180.8%	147.6%	164.3%	137.2%
gen	2	9	3	136.2%	122.4%	115.2%	156.8%
khb05250	1	10	18	354.4%	105.5%	135.2%	105.5%
lp4l	32	30	1	160.1%	146.7%	204.5%	168.7%
lseu	9	9	20	157.6%	157.6%	270.5%	137.2%
misc01	8	8	32	221.7%	221.7%	221.7%	182.2%
misc03	30	28	28	106.9%	100.0%	257.0%	199.5%
misc05	13	13	20	143.0%	143.0%	165.9%	139.8%
mod008	21	20	13	143.5%	166.4%	129.7%	110.6%
mod013	1	8	18	522.9%	160.8%	160.8%	143.2%
p0033	1	26	20	351.1%	173.2%	160.9%	195.1%
p0201	9	9	18	180.8%	180.8%	168.8%	189.8%
pipex	7	9	29	158.9%	110.8%	201.7%	125.4%
rgn	5	7	8	136.9%	150.3%	100.0%	147.1%
sentoy	11	11	16	113.2%	111.0%	145.3%	126.7%
stein27	9	10	30	128.2%	126.6%	136.0%	126.6%
truck4	3	5	13	498.2%	223.0%	126.7%	216.7%
tsp43	8	8	10	100.2%	100.2%	100.2%	100.0%
utrans.1	3	8	14	205.8%	188.9%	188.9%	140.8%
vasilis2	32	32	32	100.0%	100.0%	157.6%	144.0%
			Average	200.0%	144.1%	161.2%	146.5%
			Standard deviation	116.6%	35.4%	44.8%	31.6%

Table 4: Percentage comparison, first batch.

640kb memory, and only using values between 1 and 32.

We do better using the new prediction than using 8 constantly but only marginally better than using 10 constantly.

It is perhaps easier to compare the different values by comparing the four choices, automatic, predicted, 8 and 10 as a percentage of the true value. These calculations are presented in table 4.

It is evident from table 4 that a relatively simple data structure, like the neural network, can improve the prediction considerably. On average the automatic choice does twice as much work as is necessary while the predicted choice results in approximately 44% more work.

Problem name	Auto. value	Pred. value	Time auto.	Time pred.	Time 8	Time 10
mod010	32	30	28.60	28.57	37.93	28.29
cracpb1	32	29	63.45	69.50	333.18	196.51
vasilis_2	32	32	68.29	68.29	97.99	66.20
utrans.2	2	8	372.03	221.54	223.93	256.41
		Sum	532.4	387.9	693.0	547.4
			97.3%	70.9%	126.6%	100.0%
utrans.3	4	9	592.25	573.96	586.44	680.13
c-fat200-1	32	29	1328.07	1348.92	2124.40	2053.97
vasilis	3	18	2258.20	707.33	1161.02	723.46
vasilis_3	3	6	2423.32	1102.90	1548.16	801.77
martin	31	30	2488.12	1320.63	2379.58	2241.62
genova6xs	32	30	2651.46	2640.09	5608.61	4093.47
vpm1	14	13	4493.37	4154.15	2722.17	4260.35
san200_0.9_3	32	30	4643.80	2464.03	2029.80	4238.18
l152lav	32	29	4809.20	4970.13	8993.64	9466.21
misc07	23	21	10847.23	10544.02	14102.32	13094.43
stein45	19	18	14304.80	14204.36	17769.67	15381.07
vasilis_1	2	13	32647.07	462.30	577.54	2027.62
		Sum	83486.9	44492.8	59603.4	59062.3
			141.4%	75.3%	100.9%	100.0%

Table 5: Results for the second batch, skip factor and CPU seconds.

4.6 Extended experiments

It is perhaps a bit disheartening how well the naive choice performed on the problems in the first batch. From table 4 we note that setting the skip factor to 10 results in approximately 47% more work than necessary compared to 44% using the new prediction, but with less standard deviation. In addition, always fixing the skip factor to 10 would be a lot simpler than any other method.

To examine this finding in more detail and also to examine the performance on larger problems we selected a number of problems, mostly much larger problems than before, and solved them using the four choices, automatic, prediction, 8 and 10, for the skip factor (instead of every value between 1 and 32.) We trained the neural network on every problem in the first batch and applied it to the problems in the second batch. The results can be found in table 5.

We split the problems into small and large based on the time spent solving them using the automatic choice for the skip factor. Small problems are the ones that required less than 500 seconds and large problems are the ones that required more than 500 seconds.

Again it is perhaps easier to compare the different values by comparing the choices

Problem name	Auto. value	Pred. value	Auto. vs. 10	Pred. vs. 10	8 vs. 10	Pred. vs. auto.
mod010	32	30	101.1%	101.0%	134.1%	99.9%
cracpb1	32	29	32.3%	35.4%	169.5%	109.5%
vasilis_2	32	32	103.2%	103.2%	148.0%	100.0%
utrans.2	2	8	145.1%	86.4%	87.3%	59.5%
		Average	95.4%	81.5%	134.7%	92.2%
		Standard deviation	46.7%	31.6%	34.8%	22.3%
utrans.3	4	9	87.1%	84.4%	86.2%	96.9%
c-fat200-1	32	29	64.7%	65.7%	103.4%	101.6%
vasilis	3	18	312.1%	97.8%	160.5%	31.3%
vasilis_3	3	6	302.2%	137.6%	193.1%	45.5%
martin	31	30	111.0%	58.9%	106.2%	53.1%
genova6xs	32	30	64.8%	64.5%	137.0%	99.6%
vpm1	14	13	105.5%	97.5%	63.9%	92.5%
san200_0.9_3	32	30	109.6%	58.1%	47.9%	53.1%
l152lav	32	29	50.8%	52.5%	95.0%	103.3%
misc07	23	21	82.8%	80.5%	107.7%	97.2%
stein45	19	18	93.0%	92.3%	115.5%	99.3%
vasilis_1	2	13	1610.1%	22.8%	28.5%	1.4%
		Average	249.5%	76.1%	103.7%	72.9%
		Standard deviation	437.3%	29.1%	45.9%	34.5%
		Average	211.0%	77.4%	111.5%	77.7%
		Standard deviation	381.4%	28.8%	44.5%	32.3%

Table 6: Percentage comparison, second batch.

against a reference value. This time we do not know the true value so table 6 presents the three choices automatic, predicted and 8 as a percentage of 10. The last column is predicted vs. automatic. For comparison we also present the same calculation for the first batch of problems in table 7.

We note that using 10 and the prediction value for the skip factor for the first batch of problems, results on average in the same CPU time, see table 7. However for the problems in the second batch, table 6, in particular the large problems, there is a clear difference. Only one of the larger problems took longer to solve using the new prediction as the skip factor rather than 10, and in that case there was an improvement of almost 55% of using the prediction over using the automatic choice.

We notice that we have improved the automatic choice by 15.6% for the problems in the fist batch, and by 22.3% for the problems in the second batch. In particular, we have improved the automatic choice by 27.1% for the larger problems in the second batch. We note that none of the problems in the first batch took more than

Problem name	Auto. value	Pred. value	True value	Auto. vs. 10	Pred. vs. 10	8 vs. 10	Pred vs. auto.
bm21	5	12	29	157.8%	104.2%	112.5%	66.1%
bm23	5	15	26	135.6%	74.8%	105.1%	55.1%
egout	1	21	9	91.6%	124.9%	100.9%	136.3%
fix3	2	14	1	145.8%	104.5%	108.2%	71.7%
fxch3	2	7	19	131.8%	107.6%	119.8%	81.6%
gen	2	9	3	86.8%	78.1%	73.5%	89.9%
khb05250	1	10	18	336.0%	100.0%	128.2%	29.8%
lp4l	32	30	1	94.9%	87.0%	121.2%	91.6%
lseu	9	9	20	114.9%	114.9%	197.2%	100.0%
misc01	8	8	32	121.7%	121.7%	121.7%	100.0%
misc03	30	28	28	53.6%	50.1%	128.8%	93.6%
misc05	13	13	20	102.3%	102.3%	118.7%	100.0%
mod008	21	20	13	129.8%	150.5%	117.3%	116.0%
mod013	1	8	18	365.2%	112.3%	112.3%	30.7%
p0033	1	26	20	180.0%	88.8%	82.5%	49.3%
p0201	9	9	18	95.2%	95.2%	89.0%	100.0%
pipex	7	9	29	126.8%	88.4%	160.9%	69.7%
rgn	5	7	8	93.1%	102.2%	68.0%	109.7%
sentoy	11	11	16	89.4%	87.7%	114.7%	98.1%
stein27	9	10	30	101.2%	100.0%	107.4%	98.8%
truck4	3	5	13	229.9%	102.9%	58.4%	44.7%
tsp43	8	8	10	100.2%	100.2%	100.2%	100.0%
utrans.1	3	8	14	146.2%	134.2%	134.2%	91.8%
vasilis2	32	32	32	69.5%	69.5%	109.5%	100.0%
			Average	137.5%	100.1%	112.1%	84.4%
			Standard deviation	75.5%	21.3%	28.8%	26.9%

Table 7: Percentage comparison, first batch.

500 seconds to solve using the automatic choice and thus none of them would have qualified for class of large problems.

Problem name	Pred. value	True value	Time pred.	Time true	Time b&b
bm21	12	29	8.10	5.92	4.54
bm23	15	26	7.41	6.11	4.13
egout	21	9	31.38	21.91	***
gen	9	3	217.18	177.39	***
fix3	14	1	161.75	134.65	***
fxch3	7	19	268.23	181.78	***
khb05250	10	18	58.11	55.10	***
lp4l	30	1	87.06	59.34	129.35
lseu	9	20	47.32	30.03	***
misc01	8	32	57.86	26.10	15.15
misc03	28	28	63.21	63.21	30.17
misc05	13	20	239.33	167.32	138.57
mod008	20	13	206.85	124.32	***
mod013	8	18	18.28	11.37	12.25
p0033	26	20	4.92	2.84	***
p0201	9	18	452.13	250.13	144.55
pipex	9	29	10.97	9.90	***
rgn	7	8	76.61	50.98	72.09
sentoy	11	16	11.16	10.05	8.13
stein27	10	30	258.85	204.45	172.88
truck4	5	13	11.46	5.14	***
tsp43	8	10	273.54	272.87	335.41
utrans.1	8	14	202.68	107.27	***
vasilis2	32	32	51.80	51.80	***

Table 8: Results for the first batch, branch-and-bound.

4.7 Branch-and-bound

When examining the results for the previous experiments, we wondered how a pure branch-and-bound code might perform on these problems. For some of these problems it seemed that not using the cuts at all might be the correct strategy.

We ran MIPO in branch-and-bound mode on the problems. In the previous experiments we had not put an upper limit on the solution time, but we decided now to put an upper limit of twice the worst time previously reported. The results are summarised in tables 8 and 9. A *** indicates MIPO was unable to solve the problem in the time allotted.

We note that we were only able to solve half of the problems in the first batch using branch-and-bound. Of those problems, in nearly all the cases the branch-and-bound code outperformed the branch-and-cut algorithm using the neural network prediction for the skip factor, on average by 29%. If, however, we take into account the problems we were unable to solve, the branch-and-bound algorithm performed

Problem name	Pred. value	Time pred.	Time b&b
mod010	30	28,57	***
cracpb1	29	69,50	9,79
vasilis_2	32	68,29	***
utrans.2	8	221,54	***
utrans.3	9	573,96	***
c-fat200-1	29	1348,92	785,67
vasilis	18	707,33	322,13
vasilis_3	6	1102,90	1410,51
martin	30	1320,63	***
genova6xs	30	2640,09	1737,14
vpm1	13	4154,15	***
san200_0.9_3	30	2464,03	***
l152lav	29	4970,13	1477,50
misc07	21	10544,02	6419,14
stein45	18	14204,36	17583,35
vasilis_1	13	462,30	321,37

Table 9: Results for the second batch, branch-and-bound.

on average more than 35% worse. For the larger problems in the second batch we were able to solve two-thirds of the problems using branch-and-bound, outperforming branch-and-cut by 27%. If we take all the larger problems into account the branch-and-bound approach performed on average more than 15% worse.

These results do suggest that the interval 1 to 32 for the skip factor may be to narrow. Although it may not be feasible to expand it to all positive integers, expanding it to the interval 1 to 100, perhaps only using the even integers or every third integer might be feasible. The reason why a large number of possible values cannot be used is the time it takes to build a training database.

Problem name	Time auto.	Time pred.	Time true	T. auto. dynam.	T. pred. dynam.	Auto. dyn. vs. true	Pred. dyn. vs. true
fix3	225.72	161.75	134.65	228.67	161.75	169.8%	120.1%
lp4l	95.01	87.06	59.34	95.01	87.06	160.1%	146.7%
gen	241.52	217.18	177.39	241.52	217.18	136.2%	122.4%
rgn	69.81	76.61	50.98	58.68	64.12	115.1%	125.8%
egout	23.02	31.38	21.91	23.02	31.38	105.1%	143.2%
tsp43	273.54	273.54	272.87	255.54	260.13	93.6%	95.3%
mod008	178.36	206.85	124.32	146.44	186.5	117.8%	150.0%
truck4	25.61	11.46	5.14	22.27	11.46	433.3%	223.0%
utrans.1	220.71	202.68	107.27	139.34	197.43	129.9%	184.0%
sentoy	11.38	11.16	10.05	11.38	11.16	113.2%	111.0%
khb05250	195.25	58.11	55.10	187.47	58.11	340.2%	105.5%
mod013	59.45	18.28	11.37	33.75	17.79	296.8%	156.5%
p0201	452.13	452.13	250.13	351.08	419.99	140.4%	167.9%
fxch3	328.72	268.23	181.78	193.23	302.75	106.3%	166.5%
lseu	47.32	47.32	30.03	36.49	36.21	121.5%	120.6%
misc05	239.33	239.33	167.32	227.14	230.76	135.8%	137.9%
p0033	9.97	4.92	2.84	8.45	4.92	297.5%	173.2%
bm23	13.44	7.41	6.11	11.00	7.41	180.0%	121.3%
misc03	67.55	63.21	63.21	67.9	63.21	107.4%	100.0%
bm21	12.26	8.10	5.92	12.99	7.96	219.4%	134.5%
pipex	15.73	10.97	9.90	14.19	11.94	143.3%	120.6%
stein27	262.08	258.85	204.45	286.76	298.11	140.3%	145.8%
misc01	57.86	57.86	26.10	58.18	57.97	222.9%	222.1%
vasilis2	51.80	51.80	51.80	51.80	51.80	100.0%	100.0%
Sum/Average	3177.6	2826.2	2030.0	2762.3	2745.3	171.9%	141.4%
/Std. dev.	156.5%	139.2%	100.0%	136.1%	135.2%	87.6%	34.8%

Table 10: Results for the first batch, dynamic update.

4.8 Dynamic tuning

Based on prior knowledge that the cuts become more shallow as the solution process progresses we also investigated if it would help to re-evaluate the formula during the solution process. The inputs are all a measure of cut quality and the quality might change while the problem is being solved.

We tried both to dynamically update the automatic choice and also dynamically update the prediction. The results can be found in table 10. While updating the automatic choice dynamically shows considerable improvement, updating the predicted choice dynamically shows almost no improvement.

This venue of dynamic update seems not to be promising. One of the problems with dynamically updating the skip factor is that all time spent optimising the skip factor has to be offset by savings in the solution time. Many methods for parameter optimisation rely on the assumption that computational time is ample. In that case the cost being minimised is financial rather than time.

To make the dynamic skip factor update more accurate, experiments have to

Problem name	Auto. value	Pred. value	True value	Time auto.	Time pred.	Time true	Time 8	Time 10
bm21	5	19	29	12.26	7.12	5.92	8.74	7.77
bm23	5	15	26	13.44	8.31	6.11	10.42	9.91
egout	1	12	9	23.02	23.81	21.91	25.35	25.12
fix3	2	2	1	225.72	225.72	134.65	167.55	154.84
fxch3	2	12	19	328.72	221.19	181.78	298.66	249.35
gen	2	22	3	241.52	240.68	177.39	204.29	278.09
khb05250	1	22	18	195.25	73.68	55.10	74.49	58.11
lp4l	32	19	1	95.01	89.43	59.34	121.37	100.12
lseu	9	10	20	47.32	41.19	30.03	81.23	41.19
misc01	8	14	32	57.86	36.15	26.10	57.86	47.56
misc03	30	22	28	67.55	79.41	63.21	162.43	126.08
misc05	13	32	20	239.33	185.93	167.32	277.57	233.89
mod008	21	11	13	178.36	135.44	124.32	161.23	137.46
mod013	1	10	18	59.45	16.28	11.37	18.28	16.28
p0033	1	6	20	9.97	3.30	2.84	4.57	5.54
p0201	9	18	18	452.13	250.13	250.13	422.30	474.73
pipex	7	9	29	15.73	10.97	9.90	19.97	12.41
rgn	5	17	8	69.81	79.14	50.98	50.98	74.97
sentoy	11	16	16	11.38	10.05	10.05	14.60	12.73
stein27	9	16	30	262.08	226.36	204.45	278.06	258.85
truck4	3	5	13	25.61	11.46	5.14	6.51	11.14
tsp43	8	18	10	273.54	514.38	272.87	273.54	272.87
utrans.1	3	13	14	220.71	146.86	107.27	202.68	151.00
vasilis2	32	19	32	51.80	59.59	51.80	81.62	74.57
			Sum	3177.6	2696.6	2030.0	3024.3	2834.6
				156.5%	132.8%	100.0%	149.0%	139.6%

Table 11: Results for the first batch, other inputs.

made on the fly based on how the cuts are performing and how the solution is progressing.

4.9 Other inputs

We also experimented briefly with using different inputs than those mentioned in section 4.5. In particular we tried using the automatic value suggested by MIPO, the cut-off distance at the root and the condition number of the optimal basis of the linear relaxation.

There are some indications that the condition number of the optimal basis of the linear relaxation might play some role in the complexity of the problem. We calculated the condition number at the root node of the search tree and did the same analysis as before on both batches of problems. Due to the large range of

Problem name	Auto. value	Pred. value	True value	Auto. vs. true	Pred. vs. true	8 vs. true	10 vs. true
bm21	5	19	29	207.1%	120.3%	147.6%	131.3%
bm23	5	15	26	220.0%	136.0%	170.5%	162.2%
egout	1	12	9	105.1%	108.7%	115.7%	114.7%
fix3	2	2	1	167.6%	167.6%	124.4%	115.0%
fxch3	2	12	19	180.8%	121.7%	164.3%	137.2%
gen	2	22	3	136.2%	135.7%	115.2%	156.8%
khb05250	1	22	18	354.4%	133.7%	135.2%	105.5%
lp4l	32	19	1	160.1%	150.7%	204.5%	168.7%
lseu	9	10	20	157.6%	137.2%	270.5%	137.2%
misc01	8	14	32	221.7%	138.5%	221.7%	182.2%
misc03	30	22	28	106.9%	125.6%	257.0%	199.5%
misc05	13	32	20	143.0%	111.1%	165.9%	139.8%
mod008	21	11	13	143.5%	108.9%	129.7%	110.6%
mod013	1	10	18	522.9%	143.2%	160.8%	143.2%
p0033	1	6	20	351.1%	116.2%	160.9%	195.1%
p0201	9	18	18	180.8%	100.0%	168.8%	189.8%
pipex	7	9	29	158.9%	110.8%	201.7%	125.4%
rgn	5	17	8	136.9%	155.2%	100.0%	147.1%
sentoy	11	16	16	113.2%	100.0%	145.3%	126.7%
stein27	9	16	30	128.2%	110.7%	136.0%	126.6%
truck4	3	5	13	498.3%	223.0%	126.7%	216.7%
tsp43	8	18	10	100.3%	188.5%	100.3%	100.0%
utrans.1	3	13	14	205.8%	136.9%	188.9%	140.8%
vasilis2	32	19	32	100.0%	115.0%	157.6%	144.0%
			Average	200.0%	133.1%	161.2%	146.5%
			Standard deviation	116.6%	28.8%	44.8%	31.6%

Table 12: Percentage comparison, first batch.

values we experienced, we decided to scale the values by using the logarithm of the condition number.

The results are similar to what we had already got. The main difference is that we are able to predict on average with more accuracy what value to use for the skip factor for the smaller problems, but with less accuracy for the larger problems.

For the smaller problems, if we compare tables 4 and 12 we note that the average ratio of solution times, using the predicted value versus using the true value for the skip factor, drops from 144.1% to 133.1% and the deviation decreases from 35.4% to 28.8%. If we compare tables 7 and 15 we note that the ratio of using the predicted value versus using 10 or the automatic value decreases, but the standard deviation increases.

For the larger problems, if we compare tables 6 and 14 we note that the average ratio of solution times, using the predicted value versus using 10 for the skip factor,

Problem name	Auto. value	Pred. value	Time auto.	Time pred.	Time 8	Time 10
mod010	32	22	28.60	27.73	37.93	28.29
cracpb1	32	23	63.45	91.10	333.18	196.51
vasilis_2	32	20	68.29	62.33	97.99	66.20
utrans.2	2	12	372.03	186.77	223.93	256.41
		Sum	532.4	367.9	693.0	547.4
			97.3%	67.2%	126.6%	100.0%
utrans.3	4	14	592.25	566.98	586.44	680.13
c-fat200-1	32	32	1328.07	1328.07	2124.40	2053.97
vasilis	3	32	2258.20	497.20	1161.02	723.46
vasilis_3	3	11	2423.32	599.14	1548.16	801.77
martin	31	19	2488.12	2116.54	2379.58	2241.62
genova6xs	32	21	2651.46	3195.46	5608.61	4093.47
vpm1	14	23	4493.37	4565.72	2722.17	4260.35
san200_0.9_3	32	31	4643.80	3944.17	2029.80	4238.18
l152lav	32	21	4809.20	4575.03	8993.64	9466.21
misc07	23	25	10847.23	10089.04	14102.32	13094.43
stein45	19	23	14304.80	14158.51	17769.67	15381.07
vasilis_1	2	9	32647.07	620.87	577.54	2027.62
		Sum	83486.9	46256.7	59603.4	59062.3
			141.4%	78.3%	100.9%	100.0%

Table 13: Results for the second batch, other inputs.

remains almost constant (76.1% versus 76.0%) but the ratio, using the predicted value versus using the automatic value for the skip factor, increases from 72.9% to 77.0%, with increased deviation.

Problem name	Auto. value	Pred. value	Auto. vs. 10	Pred. vs. 10	8 vs. 10	Pred. vs. auto.
mod010	32	22	101,1%	98,0%	134,1%	97,0%
cracpb1	32	23	32,3%	46,4%	169,5%	143,6%
vasilis_2	32	20	103,2%	94,2%	148.0%	91.3%
utrans.2	2	12	145.1%	72.8%	87.3%	50.2%
		Average	95.4%	77.8%	134.7%	95.5%
		Standard deviation	46.7%	23.7%	34.8%	38.2%
utrans.3	4	14	87.1%	83.4%	86.2%	95.7%
c-fat200-1	32	32	64.7%	64.7%	103.4%	100.0%
vasilis	3	32	312.1%	68.7%	160.5%	22.0%
vasilis_3	3	11	302.2%	74.7%	193.1%	24.7%
martin	31	19	111.0%	94.4%	106.2%	85.1%
genova6xs	32	21	64.8%	78.1%	137.0%	120.5%
vpm1	14	23	105.5%	107.2%	63.9%	101.6%
san200_0.9_3	32	31	109.6%	93.1%	47.9%	84.9%
l152lav	32	21	50.8%	48.3%	95.0%	95.1%
misc07	23	25	82.8%	77.0%	107.7%	93.0%
stein45	19	23	93.0%	92.1%	115.5%	99.0%
vasilis_1	2	9	1610.1%	30.6%	28.5%	1.9%
		Average	249.5%	76.0%	103.7%	77.0%
		Standard deviation	437.3%	21.2%	45.9%	38.1%
		Average	211.0%	76.5%	111.5%	81.6%
		Standard deviation	381.4%	21.0%	44.5%	37.8%

Table 14: Percentage comparison, second batch.

Problem name	Auto. value	Pred. value	True value	Auto. vs. 10	Pred. vs. 10	8 vs. 10	Pred vs. auto.
bm21	5	19	29	157.8%	91.6%	112.5%	58.1%
bm23	5	15	26	135.6%	83.9%	105.1%	61.8%
egout	1	12	9	91.6%	94.8%	100.9%	103.4%
fix3	2	2	1	145.8%	145.8%	108.2%	100.0%
fxch3	2	12	19	131.8%	88.7%	119.8%	67.3%
gen	2	22	3	86.8%	86.5%	73.5%	99.7%
khb05250	1	22	18	336.0%	126.8%	128.2%	37.7%
lp4l	32	19	1	94.9%	89.3%	121.2%	94.1%
lseu	9	10	20	114.9%	100.0%	197.2%	87.0%
misc01	8	14	32	121.7%	76.0%	121.7%	62.5%
misc03	30	22	28	53.6%	63.0%	128.8%	117.6%
misc05	13	32	20	102.3%	79.5%	118.7%	77.7%
mod008	21	11	13	129.8%	98.5%	117.3%	75.9%
mod013	1	10	18	365.2%	100.0%	112.3%	27.4%
p0033	1	6	20	180.0%	59.6%	82.5%	33.1%
p0201	9	18	18	95.2%	52.7%	89.0%	55.3%
pipex	7	9	29	126.8%	88.4%	160.9%	69.7%
rgn	5	17	8	93.1%	105.6%	68.0%	113.4%
sentoy	11	16	16	89.4%	78.9%	114.7%	88.3%
stein27	9	16	30	101.2%	87.4%	107.4%	86.4%
truck4	3	5	13	229.9%	102.9%	58.4%	44.7%
tsp43	8	18	10	100.2%	188.5%	100.2%	188.0%
utrans.1	3	13	14	146.2%	97.3%	134.2%	66.5%
vasilis2	32	19	32	69.5%	79.9%	109.5%	115.0%
			Average	137.5%	94.4%	112.1%	80.5%
			Standard deviation	75.5%	28.1%	28.8%	34.5%

Table 15: Percentage comparison, first batch.

5 Further research

A point of further research would be to acquire more data. More data would allow a more detailed study into whether a more complex architecture would yield better results, or if this is as close as we can get.

As mentioned in section 4.5 the curve of CPU time as a function of the skip factor could sometimes be quite erratic. To compensate for that a different architecture should perhaps be used to recognise the correct value for the skip factor, using more inputs and a more complex network. Additional inputs of interest include the density of the problem, size of the problem, the angle between the cuts generated and the objective function, and the condition number of the optimal basis of the linear relaxation; and their theoretical connection to the complexity of the problem. By a more complex network we are referring to both adding additional inputs and also adding layers and hidden units.

Another idea still awaiting more data is to split the training database up into classes of problems with similar combinatorial structure. When presented with a problem for which a skip factor should be determined, perhaps the neural network should pay more attention to problems in the training database with combinatorial structure similar to the problem at hand, than treating all problems equally.

Finally, an interesting idea would to explore further dynamic tuning, taking not only global history into account as we have done in this paper, but also local history and properties of the problem at each node in the search tree.

6 Conclusion

In this paper we investigated the use of neural networks to aid in selecting a value for an important parameter in a branch-and-cut algorithm.

We showed that a relatively simple neural network can be a significant aid and better than a naive choice, in particular for large problems. The downside is that it requires a database that can be time-consuming to compile. However, once a starting database has been built, new data can always be added later on and the neural network easily re-trained.

Some questions are left unanswered for further research, such as whether a larger interval should used for the skip factor, to be able to more closely simulate a pure branch-and-bound code. Also, how it might be possible to update the skip factor dynamically. Time spent on such update would have to be offset by decrease in solution time.

We hope to have raised the issue of parameter tuning for heuristics. Choosing values for parameters are an important part of algorithms and researchers and developers can improve the performance of their algorithms out in the field by paying attention to such issues.

References

[1] Balas, E., S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming,* 58:295–324, 1993.

[2] Balas, E., S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science,* 42(9):1229–1246, September 1996.

[3] Battiti, R. First- and second-order methods for learning: Between steepest descent and Newton's Method. *Neural Computation,* 4:141–166, 1992.

[4] Battiti, R. Machine learning methods for parameter tuning in heuristics. Preliminary version, *5th DIMACS Challenge Workshop: Experimental Methodology Day,* October 1996.

[5] Bertsekas, D. P. and J. N. Tsitsiklis. *Neuro-dynamic programming,* Athena Scientific, 1996.

[6] Box, G. E. P. and N. R. Draper. *Empirical model-building and response surfaces,* John Wiley & sons, 1987.

[7] Box, G. E. P., W. G. Hunter and J. S. Hunter. *Statics for experimenters: An introduction to design, data analysis and model building,* John Wiley & sons, 1978.

[8] Dawande, M. Private communication, 1997.

[9] Fletcher, R. *Practial Methods of optimization, second edition,* John Wiley & sons, 1987.

[10] Gomory, R. An algorithm for the mixed integer problem. *Technical Report RM-2597,* The Rand Corporation, Santa Monica, CA, 1960.

[11] Hooker, J. N. Testing heuristics: We have it all wrong. *Journal of Heuristics,* 1:33–42, 1995.

[12] Luenberger, D. G. *Linear and nonlinear programming, second edition,* Addison-Wesley, 1984.

[13] Moore, A. W. *Artificial Intelligence Core, lecture notes,* 1997.

[14] Moore, A. W. and J. Schneider. Memory-based stochastic optimization. Carnegie Mellon University, PA, 1997.

[15] Padberg, M. and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review,* 33(1):60–100, March 1991.

[16] Russel, S. and P. Norvig. *Artificial intelligence: A modern approach,* Prentice Hall, 1995.

[17] Thienel, S. Inaugural-Dissertation zur Erlangung des Doktorgrades der Mathmatisch-Naturwissenschaftlichen Fakultät der Universität zu Köln. *ABACUS: A Branch-And-CUt System,* Köln 1995.

A Description of problems

A.1 First batch

BM21 Originator

Formulator

Donator to MIPLIB

Comments Small but difficult 0-1 problem.

Reference

BM23 Originator

Formulator

Donator to MIPLIB

Comments Small but difficult 0-1 problem.

Reference B. Bouvier and G. Messoumian. Programmes lineaires en variables bivalentes – algorithm de Balas, Universite de Grenoble, France, 1965.

EGOUT Originator Etienne Loute.

Formulator Laurence A. Wolsey.

Donator to MIPLIB Martin Savelsbergh.

Comments With fixed-charge network flow structure.

Reference T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation, Oper. Res. 35, No. 1, pp. 45-57, 1987.

FIX3 Originator

Formulator

Donator to MIPLIB

Comments

Reference

FXCH3 Originator

Formulator

Donator to MIPLIB

Comments Fixed-charge network flow.

Reference E. Balas, S. Ceria and G. Cornuejols. A lift and project cutting plane algorithm for mixed 0-1 program, Math. Programming, 58 295-324, 1993.

GEN Originator

Formulator Laurence A. Wolsey.

Donator to MIPLIB Martin Savelsbergh.

Comments Some knapsack constraints.

Reference T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation, Oper. Res. 35, No. 1, pp. 45-57, 1987.

KHB05250 Originator Kuhn-Hamburger.

Formulator Laurence A. Wolsey.

Donator to MIPLIB Martin Savelsbergh

Comments

Reference T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation, Oper. Res. 35, No. 1, pp. 45-57, 1987.

LP4L Originator

Formulator

Donator to MIPLIB

Comments

Reference

LSEU Originator C. E. Lemke and K. Spielberg.

Formulator Ellis L. Johnson and Uwe H. Suhl

Donator to MIPLIB John J. Forrest

Comments

Reference C. Lemke and K. Spielberg. Direct search zero-one and mixed integer programming, Oper. Res. 15, pp. 892-914, 1967.

MISC01 Originator

Formulator

Donator to MIPLIB

Comments

Reference

MISC03 Originator

Formulator

Donator to MIPLIB Greg Astfalk.

Comments

Reference

MISC05 Originator

Formulator

Donator to MIPLIB Greg Astfalk.

Comments Circuit design.

Reference

MOD008 Originator IBM France

Formulator IBM France

Donator to MIPLIB John J. Forrest.

Comments

Reference

MOD013 Originator

Formulator

Donator to MIPLIB

Comments

Reference

P0033 Originator CJP set.

Formulator

Donator to MIPLIB E. Andrew Boyd.

Comments Pure 0-1 problem.

Reference Harlan Crowder, Ellis L. Johnson and Manfred Padberg. Solving Large-Scale Zero-One Linear Programming Problems, Operations Research. Vol. 31, No. 5, September-October, 1983.

P0201 Originator CJP set.

Formulator

Donator to MIPLIB E. Andrew Boyd.

Comments Pure 0-1 problem.

Reference Harlan Crowder, Ellis L. Johnson and Manfred Padberg. Solving Large-Scale Zero-One Linear Programming Problems, Operations Research. Vol. 31, No. 5, September-October 1983.

PIPEX Originator

Formulator

Donator to MIPLIB

Comments

Reference

RGN Originator Linus E. Schrage.

Formulator Laurence A. Wolsey.

Donator to MIPLIB Martin Savelsbergh.

Comments Fixed charge problem.

Reference

SENTOY Originator

Formulator

Donator to MIPLIB

Comments

Reference S. Senju and Y. Toyoda. An Approach to Linear Programming with 0-1 Variables, Management Science 15, pp. B196-B207, 1968.

STEIN27 Originator George L. Nemhauser.

Formulator John W. Gregory.

Donator to MIPLIB E. Andrew Boyd

Comments Steiner triple formulation.

Reference

TRUCK4 Originator

Formulator

Donator to MIPLIB

Comments

Reference

TSP43 Originator

Formulator

Donator to MIPLIB

Comments 43-city asymmetric travelling salesman problem.

Reference

UTRANS.1 Originator

Formulator

Donator to MIPLIB

Comments

Reference

VASILIS2 Originator

Formulator

Donator to MIPLIB

Comments

Reference

A.2 Second batch, small problems

CRACCPB1 Originator

Formulator

Donator to MIPLIB

Comments

Reference

MOD010 Originator IBM Yorktown Heights.

Formulator IBM Yorktown Heights.

Donator to MIPLIB John J. Forrest.

Comments

Reference

UTRANS.2 Originator

Formulator

Donator to MIPLIB

Comments

Reference

VASILIS_2 Originator

Formulator

Donator to MIPLIB

Comments

Reference

A.3 Second batch, large problems

C-FAT200-1 Originator

Formulator

Donator to MIPLIB

Comments Maximum stable set problem.

Reference

GENOVA6XS Originator

Formulator

Donator to MIPLIB

Comments Set covering problem.

Reference

L152LAV Originator Harlan Crowder.

Formulator Harlan Crowder.

Donator to MIPLIB John W. Gregory.

Comments

Reference

MARTIN Originator

Formulator

Donator to MIPLIB

Comments

Reference

MISC07 Originator

Formulator

Donator to MIPLIB

Comments

Reference

SAN200_0.9_3 Originator

Formulator

Donator to MIPLIB

Comments

Reference

STEIN45 Originator George L. Nemhauser.

Formulator John W. Gregory.

Donator to MIPLIB E. Andrew Boyd.

Comments Steiner triple problem.

Reference

UTRANS.3 Originator

Formulator

Donator to MIPLIB

Comments

Reference

VASILIS Originator

Formulator

Donator to MIPLIB

Comments

Reference

VASILIS_1 Originator

Formulator

Donator to MIPLIB

Comments

Reference

VASILIS_3 Originator

Formulator

Donator to MIPLIB

Comments

Reference

VPM1 Originator

Formulator Laurence A. Wolsey.

Donator to MIPLIB Martin Savelsbergh.

Comments

Reference T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation, Oper. Res. 35, No. 1, pp. 45-57, 1987.