EngSci 760: Algorithms for Optimisation

# Assignment 1: Heuristics

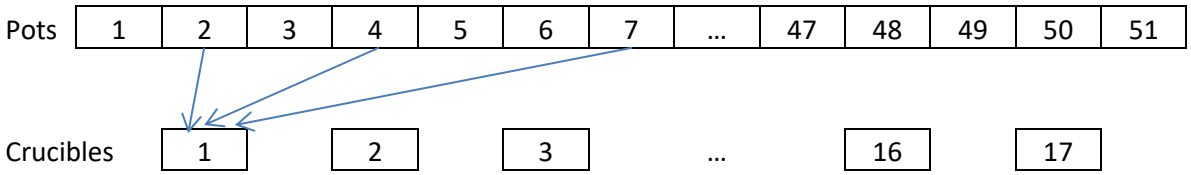
Due 27 March 2023 via Canvas Assignment 1

NZ Aluminium Smelter (NZAS) plant, Invercargil
https://commons.wikimedia.org/wiki/File:Tiwai_Point_Aluminium_Smelter.jpg

Reduction cell at Line 3. An operator positions a tapping crucible up to a cell to remove molten metal from the cell. The molten metal is then transported to the casting house for casting into ingots, t-bars or billet.
Photo provided by David Ryan.

NZ Aluminium Smelter (NZAS) Ltd faces the following problem as part of their daily production scheduling down at their Invercargil Aluminium smelter. Aluminium is produced in electrochemical pots. These pots are arranged in a number of lines, each containing 51 pots. Each line of pots needs to be emptied once a day. We will consider the problem of emptying 1 line.

The pots in a line are emptied into crucibles. Each crucible can hold the aluminium from 3 pots. Therefore, the 51 pots will be emptied into 51/3=17 crucibles. Part of a possible solution is shown below.




Each of the 17 crucibles must contain aluminium from exactly 3 pots

The quality of the aluminium in each of the crucibles is the average of the qualities of the pots used to fill the crucible. The three main quality factors are average aluminium purity, $\overline{Al}$(measured as a percentage, eg $\overline{Al}$=99.4% is good), and the average amount of steel impurity, $\overline{Fe}$(eg $\overline{Fe}$=0.01% is good) and silicon impurity $\overline{Si}$.

| Pot | Al | Fe | Si |
|---|---|---|---|
| $p$=2 | Al[2]=99.3% | Fe[2]=0.05% | Si[2]=0.025% |
| $p$ =4 | Al[4]=99.9% | Fe[4]=0.01% | Si[4]=0.005% |
| $p$=7 | Al[7]=99.99% | Fe[7]=0.08% | Si[7]=0.04% |
| Crucible $c$=1 | $\overline{Al}$ =99.73% | $\overline{Fe}$ =0.0467% | $\overline{Si}$ =0.02335% |

*Crucible Example*

In the example (right), if pots 2, 4 and 7 were emptied into the same crucible (say crucible 1), then the average percentage Al, Fe and Si in the crucible would be :

$$\overline{Al} = \frac{Al[2] + Al[4] + Al[7]}{3} = \frac{99.3 + 99.9 + 99.99}{3}$$
$$\Rightarrow \overline{Al} = 99.73\%$$

$$\overline{Fe} = \frac{Fe[2] + Fe[4] + Fe[7]}{3} = \frac{0.05 + 0.01 + 0.08}{3}$$
$$\Rightarrow \overline{Fe} = 0.0466667\%,$$

$$\overline{Si} = \frac{Si[2] + Si[4] + Si[7]}{3} = \frac{0.025 + 0.005 + 0.04}{3}$$
$$\Rightarrow \overline{Si} = 0.023333\%,$$

where Al[$p$], Fe[$p$] and Si[$p$] are the Al, Fe and Si percentage of pot $p$ respectively.

| Product Grade | Min $\overline{Al}$ | Max $\overline{Fe}$ | Max $\overline{Si}$ | Value $ |
|---|---|---|---|---|
| A | 99.1% | 0.09% | 0.09% | 19 |
| B | 99.1% | 0.08% | 0.06% | 21 |
| C | 99.1% | 0.07% | 0.07% | 27 |
| D | 99.1% | 0.06% | 0.06% | 36 |
| E | 99.2% | 0.06% | 0.05% | 41 |
| F | 99.3% | 0.05% | 0.06% | 44 |
| G | 99.4% | 0.05% | 0.04% | 46 |
| H | 99.5% | 0.05% | 0.03% | 49 |
| I | 99.7% | 0.04% | 0.02% | 55 |
| J | 99.9% | 0.03% | 0.01% | 60 |

*Sample Product Grade Classifications*

The dollar value of this crucible's contents can then be determined by seeing what grade it falls into in the 'product grade classifications' table; a sample (with random values) is shown (above). This table shows the minimum percentage $\overline{Al}$ and maximum percentage $\overline{Fe}$ and $\overline{Si}$ permitted in each grade. (The actual values to use are given in the code files... see below.) For our solution above with $\overline{Al} = 99.73\%$, $\overline{Fe}$ =0.046667% and $\overline{Si}$ =0.023333%, the best grade is:

Grade H: Min $\overline{Al}$=99.5%, Max $\overline{Fe}$=0.05%, Max $\overline{Si}$=0.03%

(We satisfy this grade's requirements because $\overline{Al} = 99.73\% \geq 99.5\%$, $\overline{Fe}$ =0.046667%≤0.05% & $\overline{Si}$ =0.023333%≤0.03%.) We do not satisfy grade I which has a stricter Max $\overline{Fe}$=0.04%.)

We will formulate this problem as follows.

Let the pots be numbered p=1, 2, …, 51, and let the matrix (i.e. 2D-array) **x**,

$$\mathbf{x}=(x_{1,1}, x_{1,2}, x_{1,3}; x_{2,1}, x_{2,2}, x_{2,3}; \ldots, x_{17,3}) = \begin{bmatrix} x_{1,1} & x_{2,1} & & x_{17,1} \\ x_{1,2} & x_{2,2} & \ldots & x_{17,2} \\ x_{1,3} & x_{2,3} & & x_{17,3} \end{bmatrix}$$

denote a solution, where $x_{cj}$, gives the 3 pots (j=1,2,3) in the c'th crucible, c=1,2,...,17, where $x_{cj} \in \{1, 2, \ldots, 51\}$. For example:

$$\mathbf{x}=(1,2,3;4,5,6;\ldots;49,50,51) = \begin{bmatrix} 1 & 4 & & 49 \\ 2 & 5 & \ldots & 50 \\ 3 & 6 & & 51 \end{bmatrix}$$

is a possible (trivial!) solution.

The quality f(**x**) of a solution **x** is given by

$$f(\mathbf{x}) = \sum_{c=1}^{17} g\left(\frac{Al[x_{c,1}] + Al[x_{c,2}] + Al[x_{c,3}]}{3}, \frac{Fe[x_{c,1}] + Fe[x_{c,2}] + Fe[x_{c,3}]}{3}, \frac{Si[x_{c,1}] + Si[x_{c,2}] + Si[x_{c,3}]}{3}\right)$$

where function g($\overline{Al}, \overline{Fe}, \overline{Si}$) gives the dollar value (from our quality lookup table) of a crucible given its average $\overline{Al}$, $\overline{Fe}$ and $\overline{Si}$ levels. We seek to maximise f(**x**), where **x** represents a valid partitioning of the pots into 17 disjoint (non-overlapping) sets of 3 pots each. We wish to solve this problem using a sensible neighbourhood search.

**Question 1**

Define a *neighbourhood rule* in words, and then give a *formal definition of a neighbour* under this rule. (There is an 'obvious' neighbourhood rule with 4 parameters that you should use; do not use a

rule giving an artificially reduced number of neighbours.) Finally, define the *neighbourhood* by defining the set of all neighbours of some solution **x** as it is defined above. This set of neighbours should be represented by defining a particular neighbour in terms of 1 or more parameters which you then give ranges for. As an example, for some other un-related problem, you might put:

N(**x**)={ **y**(**x**,a,b), a=1…200, b=1..200, a+b<300} where

$$\mathbf{y}(\mathbf{x},a,b)=(y_1, y_2, \dots, y_{300},), y_i = \begin{cases} x_b & i = a \\ x_a & i = b \\ x_i & otherwise \end{cases}.$$

Please note that you have to define the arguments for **y**(**x**, ….) that you need to specify a neighbour, and what the ranges are for each argument. Eg, you might have **y**(**x**,h,i,j,k,m), h=1,2,3, i=h, h+1, ..., 6, j=1,2,3, k=0,1 m=1,2,3,..., 100.

Your set N(**x**) should not include the same neighbouring solution more than once, nor should your neighbourhood be unnecessarily small. (Remember that, in our lecture bar seating example, we used an 'unnecessarily small' neighbourhood; you must not do this.) Your answer should give

- A definition in English of the neighbourhood rule you use to generate a neighbouring solution
- A formal definition of a function **y**(**x**,…) that produces a neighbour **y** (i.e. a neighbouring solution **y**) from **x**. (You should define the elements of y in terms of the elements of **x**, perhaps using notation similar to that in the example above. Note that the … in **y**(**x**,…) is where you insert whatever parameters you need to define a particular neighbour.)
- A definition of the set N(**x**)={**y**(x,…),…} of all neighbours of **x**, where N(**x**) contains each neighbouring solution exactly once. (The second '…' in N(**x**)={**y**(…),…} is where you specify the range of parameter values you use to generate your full neighbourhood while ensuring you generate each unique neighbouring solution exactly once.)

Your answer **must** be in terms of the **x**=($x_{cj}$, c=1, 2, …17, j=1, 2,3) given above, and use as appropriate the associated notation defined above. Do **not** invent your own notation unless additional values/definitions are required.

**Question 2**

We wish to determine the change in objective function f(**y**)-f(**x**) when moving to some neighbour **y**=**y**(**x**,…). This calculation should be efficient; specifically, it should not recalculate values associated with any crucibles that are not impacted by the change in solution, nor should it compute the total profit from scratch each time.

To be efficient, you will need to define a single extra intermediate value for each crucible so that values are not unnecessarily recalculated when evaluating the objective change. (Calculating the value of a crucible is a slow operation, so we don't want to do lots of these calculations if we can avoid them.) The objective function calculation, and the calculation of a change in the objective function, will use these intermediate values. (An argument can be made that we should store more than one intermediate value for each crucible. However, with only 3 pots per crucible, an implementation with just one intermediate value is an acceptable trade-off between efficiency and simplicity.)

(a) Define in words these new intermediate values (one per crucible).

(b) Give a next ascent 'sweep' algorithm (in pseudo code) that checks each neighbouring solution y(**x**,...) in turn to see if it is a better than **x** (using your intermediate values in the calculations), and performs the required updates to the current solution and intermediate terms if this neighbour is indeed better. Your sweep algorithm should be designed so that if **x** is a local optimum, then your algorithm considers each neighbour exactly once. (This sweep algorithm will form the core of your iterative local search code that ascends to a local maximum; you do not need to show this full iterative local search code here.) An example of such an algorithm was given in the lecture notes for our TSP example.

Your answers must be in terms of the notation given above; it must use the *g()* function defined above, and your intermediate values, and must not refer to procedures in the computer code (see below).

**Question 3**

Using the Python code framework provided (or your own implementation of a similar framework in another language such as C, C++, C#, Julia, Matlab, …), write a *fast reasonably efficient* local search program implementing both *next-ascent* and *steepest-ascent* with random restarts to solve this problem using the local search rule and associated formulae defined by your previous answers. (Note that the code files provided include data for the 'Default' and 'Small' test problems you will solve using your code.) Your program should contain and use the functions provided in the starter code. It should be efficient in that it considers each neighbouring solution once per sweep of the neighbourhood, and performs next and steepest ascent as described in the course notes. (You will lose marks if your code "resets" unnecessarily, such as restarting a neighbourhood sweep whenever you change the solution.) The starting code and full problem data is available on Canvas.

**Task 3A Code: NextAscentToLocalMax**(x, ... )

Given a starting solution x, test (and accept where better) all neighbouring solutions in a **next ascent** approach, ie repeatedly sweep the complete neighbourhood, accepting all improvements that are found. (If no improvements are found, this will explore every neighbouring solution of x exactly once.) Be sure to repeat the sweep process until the current solution is proven to be a local maximum (i.e. all neighbours of **x** have been tested exactly once and shown to be no better than **x**).
Your code should be efficient so that when you have reached a local optimum, you do not evaluate each neighbour more than once to then prove that you are at a local optimum.

You should hand in:

- The code you write for this task (including any new functions you add and use)

**Task 3B Code: SteepestAscentToLocalMax**(x, ... )

As above, but using steepest ascent.

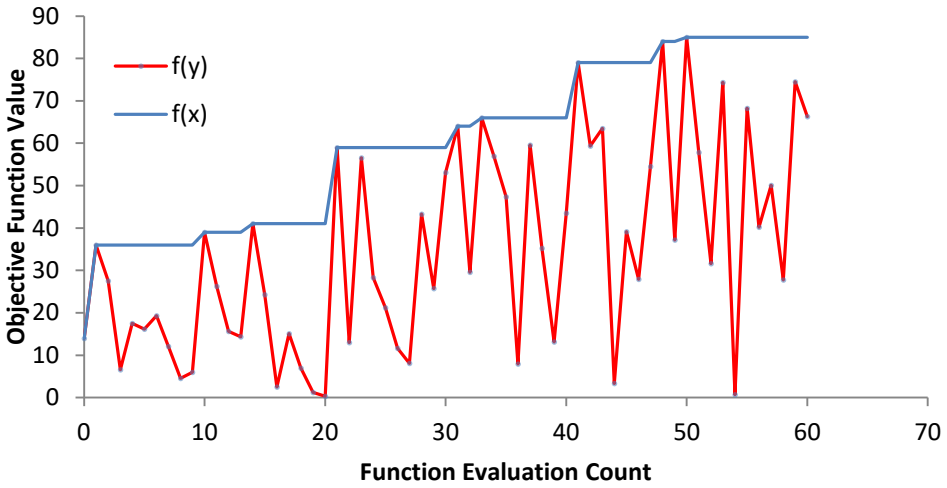



*Figure 1A: Sample representative plot for 1 run of next ascent finishing when a local maximum is found*

**Task 3C Code: Plotting of Next Ascent for the 'Small' Problem Data**

We wish to create a plot that shows us how Next Ascent works, similar to that in Figure 1A above. This plot will have "objective function value" on the y axis and 'objective function evaluation count' on the x axis. The figure will contain two data sequences. The first of these, labelled f(x) above, is the objective function value of the best solution found so far. The second data sequence, f(y), has a point for every solution evaluated (including neighbours you reject) during one full run until the search stops at a local optimum.

- To create this plot, modify the **NextAscentToLocalMax** function to record the objective function value for the current best solution and for **every** neighbour that is evaluated during the search.
- As detailed above, your plot should show both the objective f(**x**) of the current solution **x**, and the objective f(**y**) of each neighbour **y** of **x** that is evaluated.
- Your plot does not need to include a legend
- You should show f(**x**) at each step, even if it hasn't changed.
- You should start from the 'trivial' solution x=(1,2,3; 4,5,6; …).
- You should solve the "**small**" problem instance (as defined in the code).
- You can create the plot in another program (eg Excel), or produce the plot directly from your code.

You should hand in:

- An A4 landscape print of the plot

**Task 3D Code: Plotting of Steepest Ascent for the 'Small' Problem**

Repeat the above task for **steepest ascent** to produce a plot similar to Figure 1B below. Note that the current best solution objective, f(**x**), doesn't change in the plot until the start of the next full neighbourhood sweep.

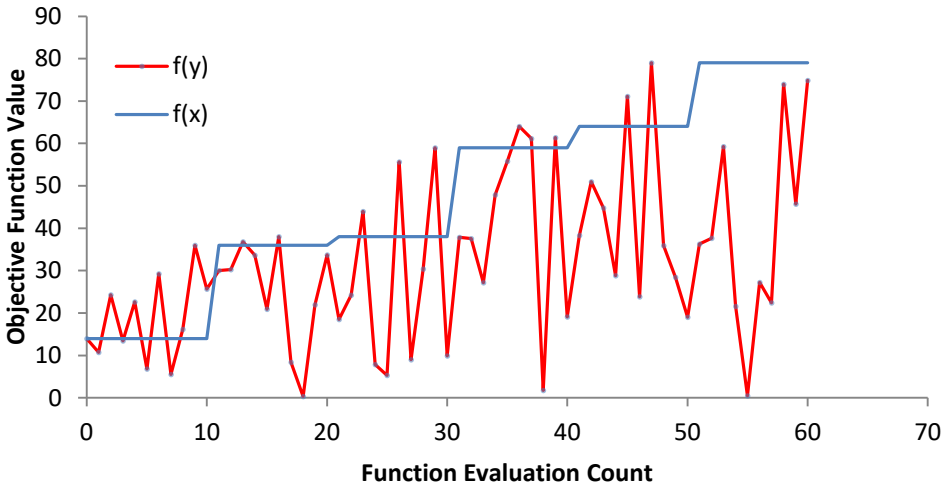



*Figure 1B: Sample representative plot for 1 run of steepest ascent finishing when a local maximum is found*

You may want to comment out (keeping visible for marking) the code changes made for this plotting so the changes don't slow down the next task.

**Task 3E Code & Plots: Repeated Next Ascents for the "Default" Problem**

Write a function **DoRepeatedNextAscents**() to do *n* repeated ascents, each starting from a random starting solution, for the "default" problem instance. Your code should create a graph showing the objective values (y-axis) against run time (x-axis). The graph should show the objective function values of each local optimum found and the best local optimum found up to that point. (This is a different plot to the ones above. Do **not** show values for intermediate solutions that are not local optima; doing so will lose you marks.) Your code should stop after exploring *n* local optima.

You should hand in:

- Your **DoRepeatedNextAscents** code, and any other functions you create for this. (Your code can include the 'spread' changes detailed next, as long as these are clearly commented.)
- The best solution found (output using **view_soln**()) for the "default" problem data after at least n=200 random restarts.
- An A4 landscape plot of the associated plot

**Task 3E Code & Plots: Repeated Steepest Ascents for the "Default" Problem**
Repeat the previous task for steepest ascent with a function **DoRepeatedSteepestAscents**().

**Question 4:** By comparing the next- and steepest-ascent plots, what can you say (if anything) about the relative efficiency of next and steepest ascent for this problem based on these (very limited) experiments?

**Question 5**
The following questions do not need to be coded up.

(a) Would you expect this problem's objective function to have lots of plateaus? Briefly explain why or why not.

(b) Give a precise definition (using words) and a formal mathematical definition (using the notation defined in this assignment, plus any new notation you need) of a new crucible value function $g'(\overline{Al}, \overline{Fe}, \overline{Si})$ that could be used instead of $g()$ to add a slope to any such plateaus. (You do not need to code up this change.) Carefully explain how you would expect your change to improve the search. Your discussion should demonstrate, using an actual example giving the objective function value changes associated with a neighbouring solution, how the search now prefers solutions with characteristics that you think (and argue in your answer) are desirable. Be sure to define all your notation precisely. You do NOT need to code up this modelling change.

**Question 6**
The smelter doesn't like mixing pots that are a long way from each other. They measure this using "spread", where the spread of the $c$'th crucible containing pots ($x_{c1}$,$x_{c2}$,$x_{c3}$) is given by:

$$s_c=\max(x_{c1},x_{c2},x_{c3})-\min(x_{c1},x_{c2},x_{c3}).$$

For example, a crucible $c$ containing pots (1,5,9) has a spread $s_c$=9-1=8.

We want to let the user set a maximum 'spread' $s$ of pots in a crucible. We will then let our algorithm generate solutions with too high a spread, but we will penalize the profit of those crucibles with too large a spread. We want our penalty function to guide the search towards feasible solutions.

Define a new mathematical function $g''(\overline{Al}, \overline{Fe}, \overline{Si}, x_{c1}, x_{c2}, x_{c3}, s)$ for this problem that calculates a penalized value of a crucible $c$ for some given maximum spread $s$. Your answer should be in terms of a mathematical formula (and the $g()$ function).

*Hint: Your penalty function should guide the search towards solutions that satisfy the spread requirement.*

**Task 6: Next Ascent with Maximum Spread of 6, 8, 11 for 'Default' Problem Data**
Modify your next-ascent code (including function arguments as required) and any supporting functions to implement this 'spread' version of the problem. (Make it clear via comments in your code what changes have been made to implement spread, or create new functions for the spread functionality.) You can also comment out any code used to produce plots; plots are not needed for this question.

You should hand in:

- Your modified **DoRepeatedNextAscents** code (or a new version of this for spread), and any other functions you create/modify for this.
- The best solutions found (output using **view_soln**()) for the "default" problem data using at least n=200 random restarts for spreads of 6, 8 and 11.

**Hand in Instructions**
You should hand in your question answers, code and plots via the Canvas electronic submission process. You should upload your answers as a single PDF (either printed or scanned hand-written pages are fine) including your plots and code.

**Academic Integrity and Copyright – No Public Posting of Your Code or Answers**
This assignment, and the sample code, is copyright. Neither of these may be made publically available in any form. You are NOT permitted to post this assignment or the sample code, nor any code derived from this (including your assignment answers), in any public forum (which includes a public listing on Github, Gitlab, or similar web sites). It is a violation of the University of Auckland rules on Academic Integrity to make your answers to this assignment publicly available.

**Prize** A prize (including Aluminium from the smelter) will be awarded for the best solution(s) found for the spread and no-spread cases submitted in your assignments. You can run for as long as you like to produce the answers. You do not need to do anything to enter the competition; just make sure the solutions you submit to the tasks above are the very best you can find.

*Note: This assignment is inspired by work undertaken by Professor Ryan with NZAS.*

## New Zealand Aluminium Smelters

*Source: https://en.wikipedia.org/wiki/New_Zealand_Aluminium_Smelters*

**History:** The smelter, located at Tiwai Point on the southern coast of the South Island, was opened in 1971 after a joint venture construction (50% Comalco, 25% Sumitomo Chemical Company and 25% Showa Denko KK).[1] Comalco later purchased Showa Denko's share in April 1986.[2] It is one of the 20 largest aluminium smelters in the world and, according to Rio Tinto Alcan, it provides NZ$3.65 billion worth of economic benefit to the New Zealand economy.[3] It produces the world's highest purity primary (i.e. directly refined made from alumina ore) aluminium. The ore is mostly imported from Australia, while the finished product mostly goes to Japan.[3]

**Facility:** The smelter currently consists of 3 lines of P69 technology cells, with 208 cells each (i.e. 624 total), and one line of 48 CD200 technology cells. In 2011 the smelter produced a record amount of aluminium, 354,030 saleable tonnes. The previous record production was 352,976 tonnes in 2007.[4] In 2009, the smelter's production was 271,902 tonnes of aluminium and that year it employed approximately 750 full-time personnel and 120 contractors.[3] The third P69 Line was built in the early 1980s as part of Muldoon government's "Think Big" projects.

**Electricity:** The smelter uses up to 610MW of electricity which is mostly supplied by the hydroelectric Manapouri Power Station. The perceived reliability of power from Manapouri played a major role in the choice of building the aluminium smelter in Southland,[3] with both the power plant and the smelter having been constructed as a joint project. The facility is the largest electricity consumer in New Zealand, and uses approximately one third of the total power of the South Island and 15% of the total power countrywide.

The electricity contract with Meridian Energy (and its predecessors) has long been criticised as being secret and effectively a tax-payer funded subsidy (or corporate welfare) for a large transnational corporation.[5] In 2007 New Zealand Aluminium Smelters negotiated a new contract for electricity supply with Meridian Energy for the continuous supply of 572 megawatts for the period 2013 to 2030.[6] Prior to the contract negotiation, New Zealand Aluminium Smelters had said that it could obtain electricity from a coal-fired station at less than 7 cents per kilowatt hour.[7] In 2007, the new contract price was estimated to be 4.7 cents per kilowatt hour or one quarter of the price charged to the domestic consumer.[5][7]

### References


1. Fox, Aaron (2001). "The power game : the development of the Manapouri-Tiwai Point electro-industrial complex, 1904-1969". *Thesis, Doctor of Philosophy*. University of Otago. Retrieved 1 January 2013.
2. Lind, Clive (1996). *The People and the Power*. Invercargill: New Zealand Aluminium Smelters. p. 248.
3. "New Zealand Aluminium Smelters Limited". Rio Tinto Alcan. 2011. Retrieved 25 August 2011.
4. "Aluminium record set in 2011". *Radio New Zealand*. 2 February 2012. Retrieved 10 April 2013.
5. "Tiwai Point Smelter: Pull The Plug On Corporate Welfare" (Press release). CAFCA. 3 October 2007. Retrieved 3 May 2012.
6. Bennett, Adam (3 October 2007). "Meridian boss hails deal with smelter". *New Zealand Herald*. Archived from the original on 25 August 2011. Retrieved 24 August 2011.
7. Fairfax NZ News (2 October 2007). "Bluff aluminium smelter and Meridian in secret deal". *Stuff*. Retrieved 20 September 2012. The new contract - understood to be about 4.7c a unit