Feasibility of Establishing a Northern Western Australian Beef Abattoir

A facility location problem

Rodolfo García-Flores

MATHEMATICS, INFORMATICS AND STATISTICS
www.csiro.au

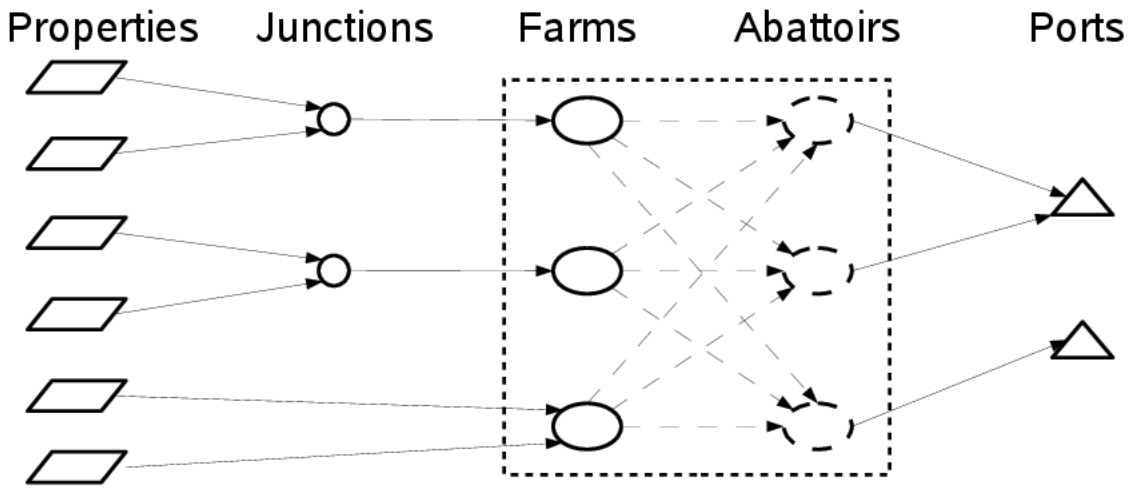
Part I

Thus spoke the master programmer:

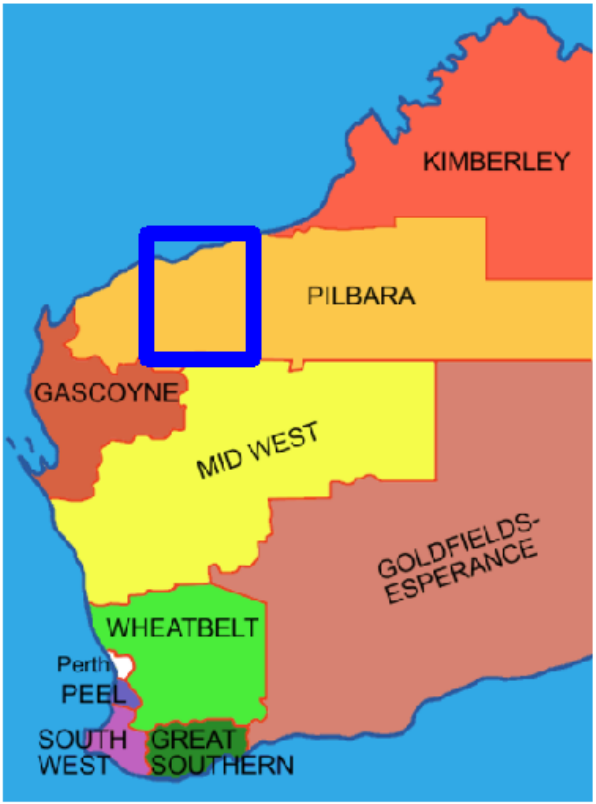
> *"Without the wind, the grass does not move. Without software, hardware is useless."*
>
> The Tao of Programming, book 8

The Northern Australian Beef Supply Chain

Work presented in the 25th European Conference on Operational Research, Vilnius, 8-11 July 2012.

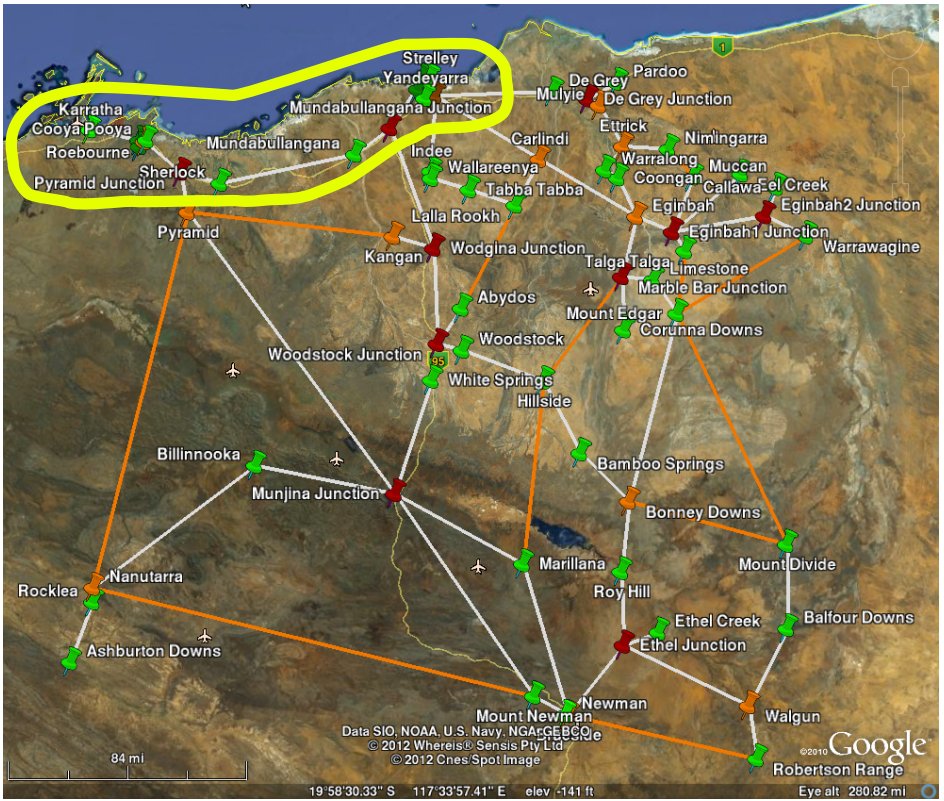


Western Australia

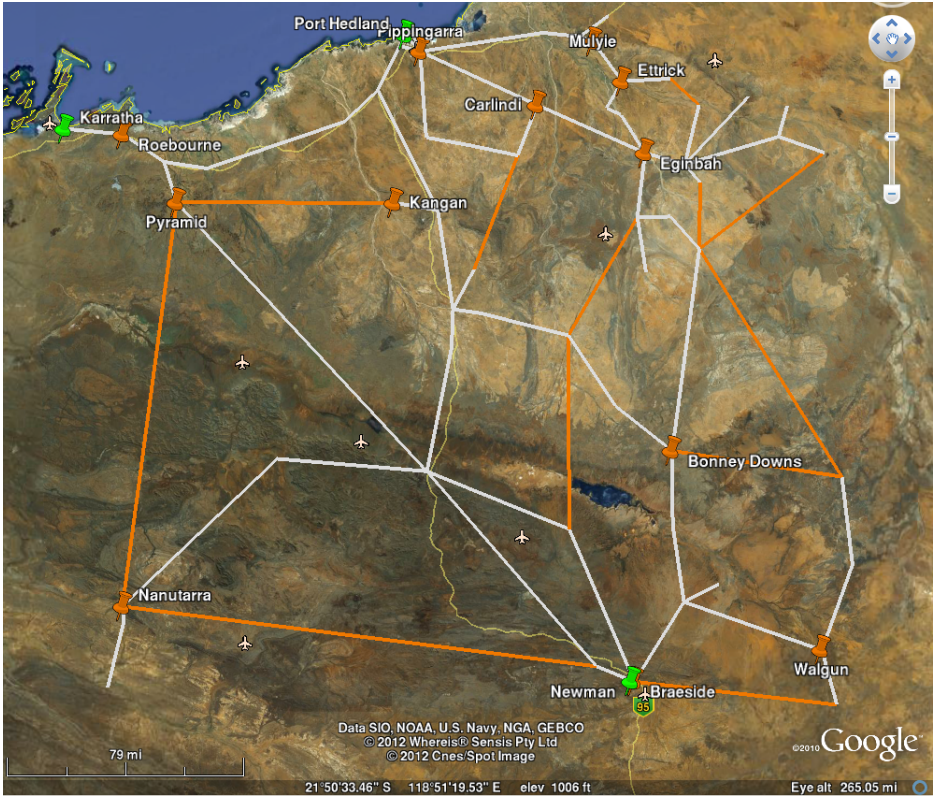


The highlighted region covers an area of 180000 km^2.

The Pilbara Beef Supply Chain



The Pilbara Beef Supply Chain



A Two-Stage Model

1. Solve the transshipment problem for the whole time horizon T (three years) to determine the number of truckloads sent from the properties qsf_{ijt} and the inventories in farms q_{it}.
2. Calculate the number of truckloads that must be transported from each farm to the abattoirs as

$$DFA_{it} = qsf_{ii,t-\tau,t} \quad \forall i \in \{A \cup F\} .$$

3. Calculate the transportation cost from farms to abattoirs $TC'_{ij} = TC_{ijt}DFA_{it}$.
4. Solve the facility location problem using the transportation cost calculated in step 2 in the transshipment objective function.

Data and parameters from Strategic Design and Development and Meateng Pty Ltd [2010] and Meateng Pty Ltd [2012].

Mathematical formulation of the transshipment problem (1)

Let qsf_{ijtu} the transported amount in truckloads and q_{it} the inventory.

Minimise

$$\sum_{(i,j)\in L_{SF}} \sum_{t\in T} \sum_{u\in T} TC_{ijtu} qsf_{ijtu} \ ,$$

Subject to:

$$\sum_{u\in T} \sum_{j\in\{S\cup F\}} ASF_{jit} qsf_{jiut} - \sum_{u\in T} \sum_{j\in\{S\cup F\}} ASF_{ijt} qsf_{ijtu} = \begin{cases} DSF_{it} & \text{if } \sum_{j\in\{S\cup F\}} ASF_{ijt} > 0 \\ 0 & \text{otherwise} \end{cases}$$

for all properties and periods.

Mathematical formulation of the transshipment problem (2)

For agistment farms,

$$\sum_{u \in T} \sum_{j \in \{S \cup F\}} ASF_{jit} qsf_{jiut} - \sum_{u \in T} \sum_{j \in \{S \cup F\}} ASF_{ijt} qsf_{ijtu} = - DSF_{it} + \quad q_{it} \quad \forall i \in F,\ \forall t \in T\ .$$

The agistment constraints are

$$\sum_{j \in S} qsf_{jitt} = qsf_{iit,t+\tau} \quad \forall i \in F,\ \forall t \in T\ ,$$

where τ is the agistment period. The farms' storage capacities are expressed as

$$q_{it} \leq FK_i \quad \forall i \in F,\ \forall t \in T\ .$$

Mathematical formulation of the facility location/network design problem (1)

Assumes that the demand of the nodes selected as abattoirs is served by a super-node [Melkote and Daskin, 2001a,b]. Let

- z_i indicate if i is an abattoir,
- xfa_{ij} indicate if a road segment is open,
- w_i^k the fraction of demand of k served by node i, and
- yfa_{ij}^k the fraction of demand of commodity k that flows on link $(i,j) \in L_{FA}$.

Minimise

$$\sum_{(i,j)\in L_{FA}} \sum_{t\in T} \sum_{k\in K_{FA}, k\neq i} (TC'_{ijt} yfa_{ij}^k + TC'_{jit} yfa_{ji}^k)$$

Mathematical formulation of the facility location/network design problem (2)

Subject to:

$$z_i + \sum_{j \in \{F \cup A\}} x f a_{ij} = 1 \quad \forall i \in \{F \cup A\} \ ,$$

that is, there are no outbound links transporting livestock from the sites chosen to be abattoirs.

$$z_k + \sum_{i \in \{F \cup A\}, i \neq k} w_i^k = 1 \quad \forall k \in \{F \cup A\}, \ \forall t \in T \ ,$$

which says that the demand of all other nodes that are not abattoirs is supplied by the abattoirs.

Mathematical formulation of the facility location/network design problem (3)

Conservation constraints:

$$xfa_{ki} + \sum_{j \in \{F \cup A\}} yfa^k_{ji} = \sum_{j \in \{F \cup A\}} yfa^k_{ij} + w^k_i$$
$$\forall i, k \in \{F \cup A\},\ i \neq k,\ \forall (k,i) \in L_{FA},\ \forall t \in T\ ,$$
$$\sum_{j \in \{F \cup A\}} yfa^k_{ji} = \sum_{j \in \{F \cup A\}} yfa^k_{ij} + w^k_i$$
$$\forall i, k \in \{F \cup A\},\ i \neq k,\ \forall (k,i) \notin L_{FA},\ \forall t \in T\ .$$

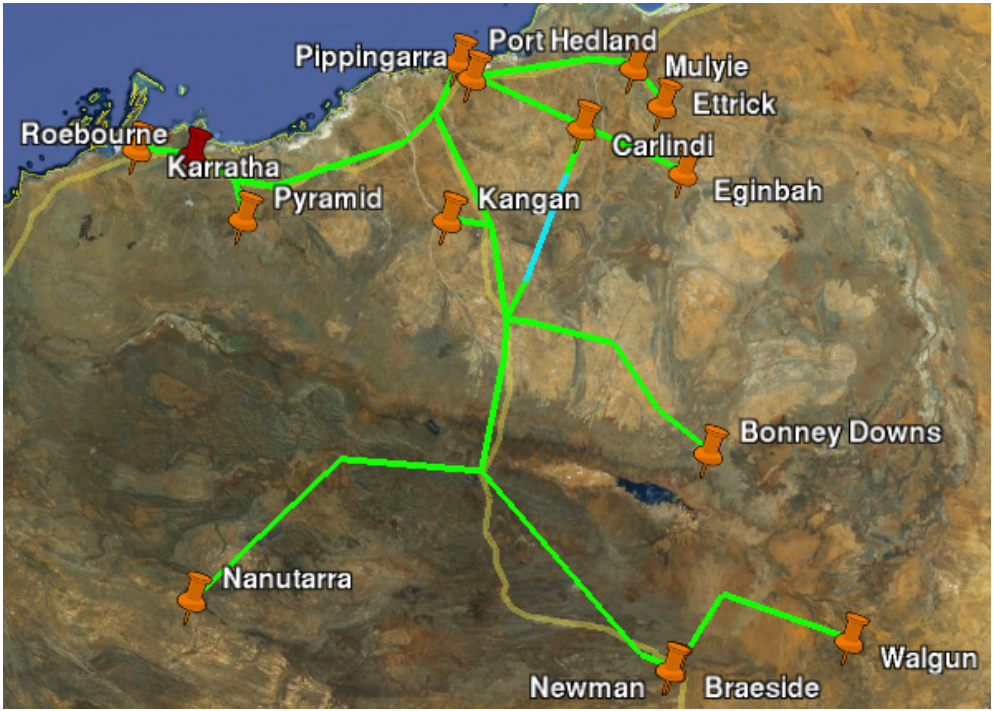
Flow is only permitted in links that are part of the transportation design,

$$yfa^k_{ij} \leq xfa_{ij} \quad \forall (i,j) \in L_{FA},\ \forall k \in \{F \cup A\}, i \neq k,\ \forall t \in T\ ,$$

and demand from a farm is served by an abattoir node,

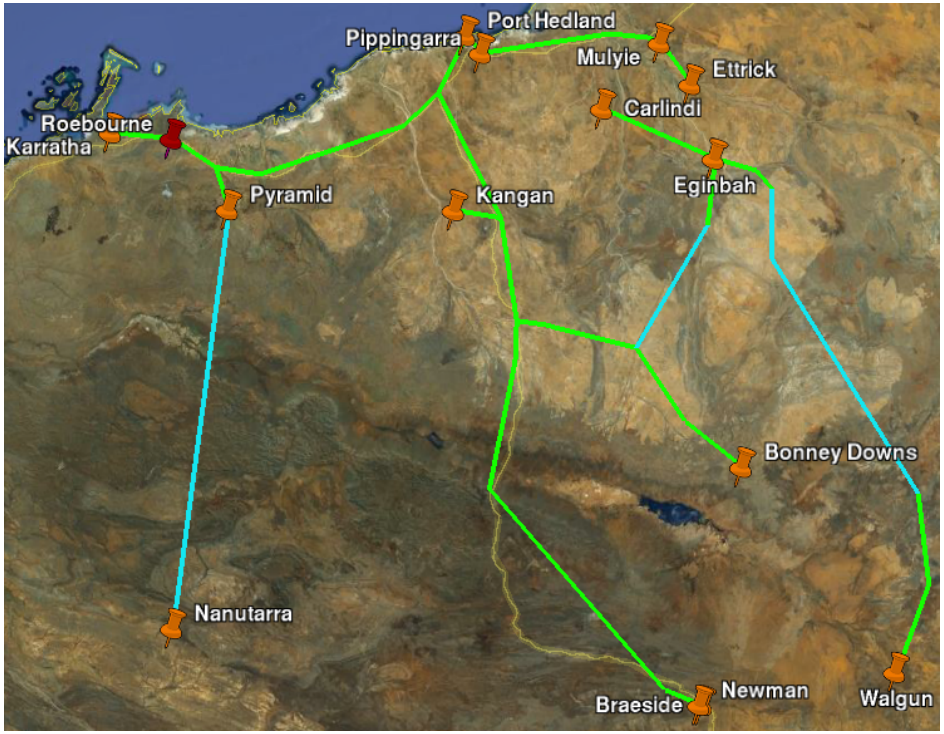
$$w^k_i \leq z_i \quad \forall i, k \in \{F \cup A\},\ i \neq k,\ \forall t \in T\ .$$

Supply Chain Design with a Budget of $50.0M



Transportation cost = AUD 8.75e+07.

Supply Chain Design with a Budget of $93.3M



Transportation cost = AUD 4.99e+07.

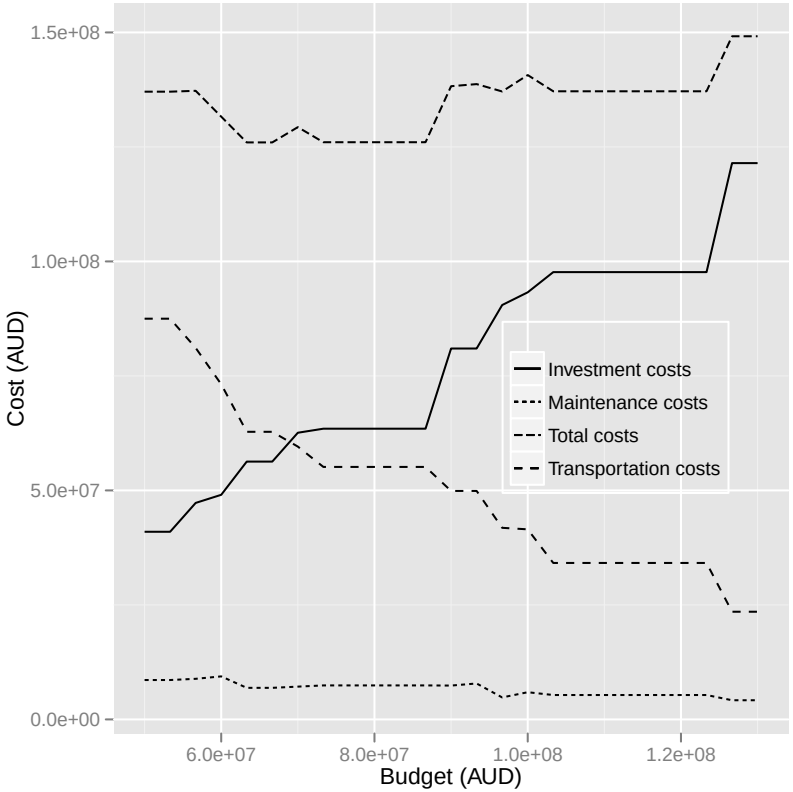
Supply Chain Design with a Budget of $100.0M

Transportation cost = AUD 4.15e+07.

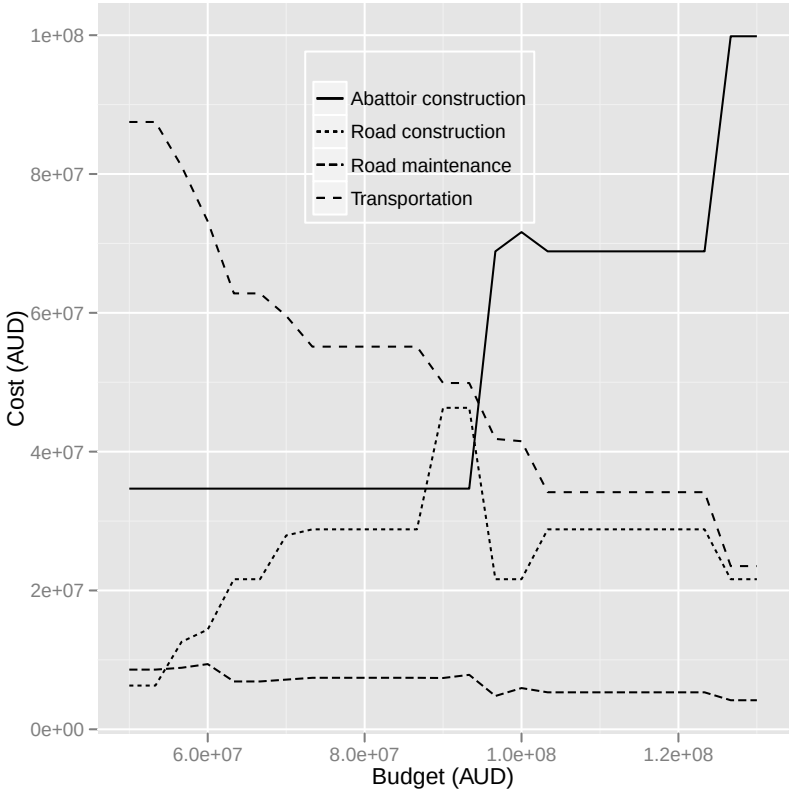
Supply Chain Design with a Budget of $130.0M

Transportation cost = AUD 2.35e+07.

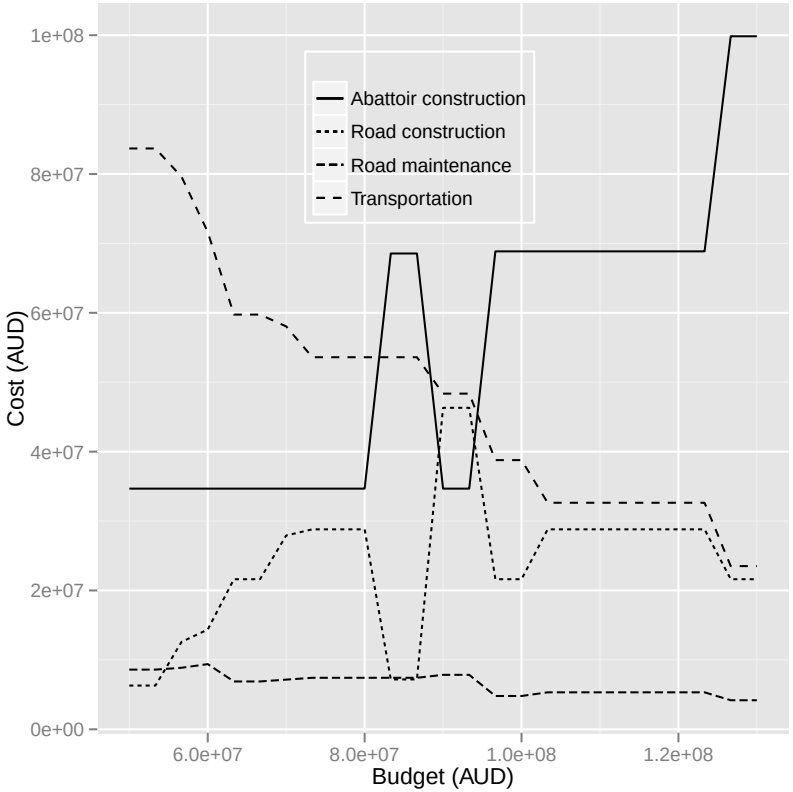
Results – Total Costs



Results – Cost Breakdown



Results – Cost Breakdown for Affected Network



References

Meateng Pty Ltd. Evaluating the commercial viability of a northern outback Queensland meat processing facility. Technical Report 136476698, Meateng Pty Ltd, 8 Montrose St., Hawthorn east, VIC 3123 Australia, February 2012.

S Melkote and MS Daskin. An integrated model of facility location and transportation network design. *Transportation Design Part A*, 35:515–538, 2001a.

S Melkote and MS Daskin. Capacitated facility location/network design problems. *European Journal of Operational Research*, 129:481–495, 2001b.

Strategic Design and Development and Meateng Pty Ltd. Feasibility of establishing a northern Australian beef abattoir. Technical Report RIRDC 10/214, Australian Government – Rural Industries Research and Development Corporation, November 2010.

Photographs taken from *http://thegreyroamer.blogspot.com.au/2011/11/murchison.html*

The rest of this presentation based on the tips given in Hunt, A. and Thomas, D., *The Pragmatic Programmer*, Addison Wesley, 1999.

Part II

Thus spoke the master programmer:

> *"After three days without programming, life becomes meaningless."*
>
> The Tao of Programming, book 2

16. Prototype to learn

- This is just a prototype, but it does all that it is supposed to do.
- It is meant to leverage on experience of previous projects.
- *Clarity* is the absolute priority.
- What you learn, you learn by doing:
  1. A new language,
  2. to document properly,
  3. to try new ideas,
  4. to solve a new problem.
- An experiment in "stone soup development".

4. Don’t live with broken windows

“Windows” that have broken in the past:

1. Dimensional analysis (units),
2. Naming of variables and functions,
3. Documentation,
4. Poor readability,
5. Respect to engineering conventions,
6. Code now, package for deployment later,
7. Project structure and orthogonal components.
8. ... and all practices that tend to cause problems later.

4. Don’t live with broken windows

```
(defvar g     SI/GRAM              "A gram as unit.")
(defvar kg    (SI/KILO SI/GRAM)    "A kilogram as unit.")
...

(defvar
  all-units (filter #(instance?
                       javax.measure.unit.Unit (deref %))
                    (vals (ns-publics *ns*)))
  "A variable with all the units with an interface
          in this namespace.")

(defn
  #^{:test ...}
  u "..."
  [magnitude dimension]
  (let [a-val (if (or
   (= (class magnitude) java.lang.Integer)
   (= (class magnitude) java.lang.Long))
(.doubleValue magnitude) magnitude)]
  (org.jscience.physics.amount.Amount/valueOf a-val dimension)))
```

4. Don’t live with broken windows

```
(defn
  #^{:test (fn []
     (let [var-1 (u 100 m)
   var-2 (u 2   m)
   var-3 (u 3   m)]
       (assert (.equals (add var-1) var-1))
       (assert (.equals (add var-1 var-2 var-3)
                        (apply add (list var-1 var-2 var-3))))))}
  add
  "The sum of scalar dimensional magnitudes in recursive form."
  ([]
     Amount/ZERO)
  ([term & other-terms]
     (loop [partial-term term remaining other-terms]
       (if (nil? remaining)
 partial-term
 (recur (.plus partial-term (first remaining)) (next remaining))))))
```

67. English is just a programming language

Writing code *IS* writing, which means:

1. Code is meant to *communicate* an idea.
2. It is an incremental process.
3. Variable and function names must be meaningful.
4. What you write remains there, staring back at you, so
5. it must be reviewed by someone else.

67. English is just a programming language

Writing code *IS* writing, which means:

1. Code is meant to *communicate* an idea.
2. It is an incremental process.
3. Variable and function names must be meaningful.
4. What you write remains there, staring back at you, so
5. it must be reviewed by someone else.

"Though a program be but three lines long, someday it will have to be maintained." Tao, book 5.

Part III

Thus spoke the master programmer:

> *"A well-written program is its own heaven; a poorly-written program is its own hell."*
>
> The Tao of Programming, book 4

17. Program close to the problem domain

```
(defn
  set-budget-constraint
    "$\\sum_{i \\in N} f_i z_i + \\sum_{(i,j) \\in L} c_{ij} xfa_{ij} \\leq B$.
     The identifier of these constraints in the LP is BUD."
  [input-data all-variables]
  (log "Setting up budget constraint and checking dimensional
        consistency.")
  (let [valid-FF-links-info (get-valid-FF-links input-data)
z                   (filter-variables all-variables abattoir-prefix)
xfa                 (filter-variables all-variables fa-link-prefix)
B                   (get-total-budget input-data)
c                   (partial get-road-investment-cost
                             input-data valid-FF-links-info)
f                  #(get-node-cost-of-opening input-data
                             (first (:set-elements %)))
sum-over            (partial get-ordered-coefficients
                             all-variables (.getUnit B))]
    (struct-map constraint-description
      :left-hand-side   (do (sum-over (list #(f %) z
                                            #(c %) xfa)))
      :relation         :<=
      :right-hand-side  B
      :index            (swap! constraint-counter inc)
      :name             (str "BUD" @constraint-counter))))
```

Should read $\sum_{i \in N} f_i z_i + \sum_{(i,j) \in L} c_{ij} x f a_{ij} \leq B$.

68. Build documentation in, don't bolt it on.

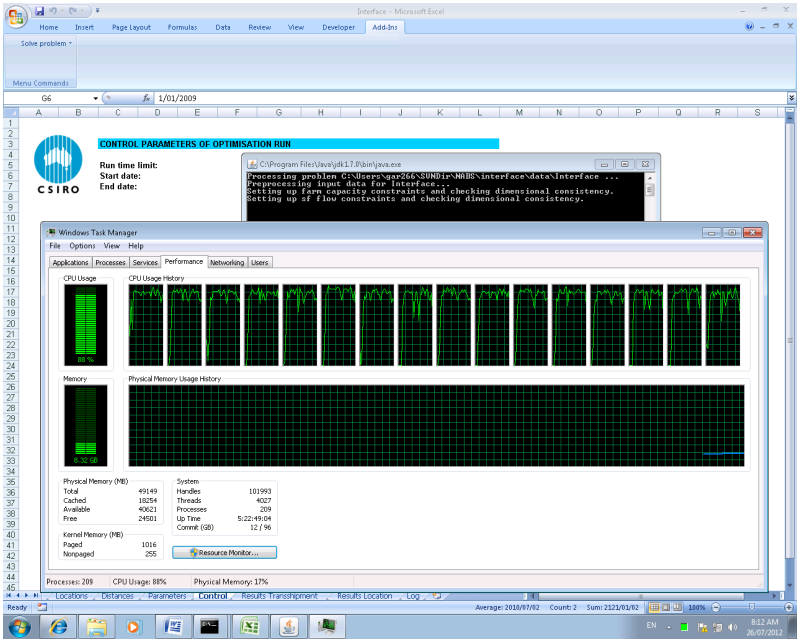
```
(defstruct action :flag :doc :function)
(def get-help-string #(str "\t" (:flag %) " " (:doc %)))

(defvar exec-lpsolve
  (struct-map action
    :flag "-xlpsolve"
    :doc "[project name] Project file (i.e., common root to CSV
          files)\n\t\tto solve with lpsolve."
    :function `execute-lpsolve)
  (str "The execution action of the application jar executable.
            Use:\n " (get-help-string exec-lpsolve)))

...
```

So there is no need to repeat the function descriptions:

```
(defvar help-string
  (str " Valid flags are (items in square
          brackets are required):\n\n"
       (apply str (interpose
             "\n" (map get-help-string all-actions))) "\n")
  "This is the full documentation and shell printout of
       \\verb|application| with no arguments.")
```

41. Always design for concurrency

It is much easier to replace `map` by `pmap` in Clojure than adding OpenMPI's `#pragma`s in C++.



20. Keep knowledge in plain text

- Input and output are CSV.
- Input and output have a GoogleEarth KML representation.
- Input and output are Clojure code.
- Progress logs and test outputs are obviously in text format.
- LP solvers have a standard representation of problems in text format.
- Problem and solution can be parsed in Clojure because code is data, but ...

Part IV

Thus spoke the master programmer:

> *"When the program is being tested, it is too late to make design changes."*
>
> The Tao of Programming, book 3

48. Design to test

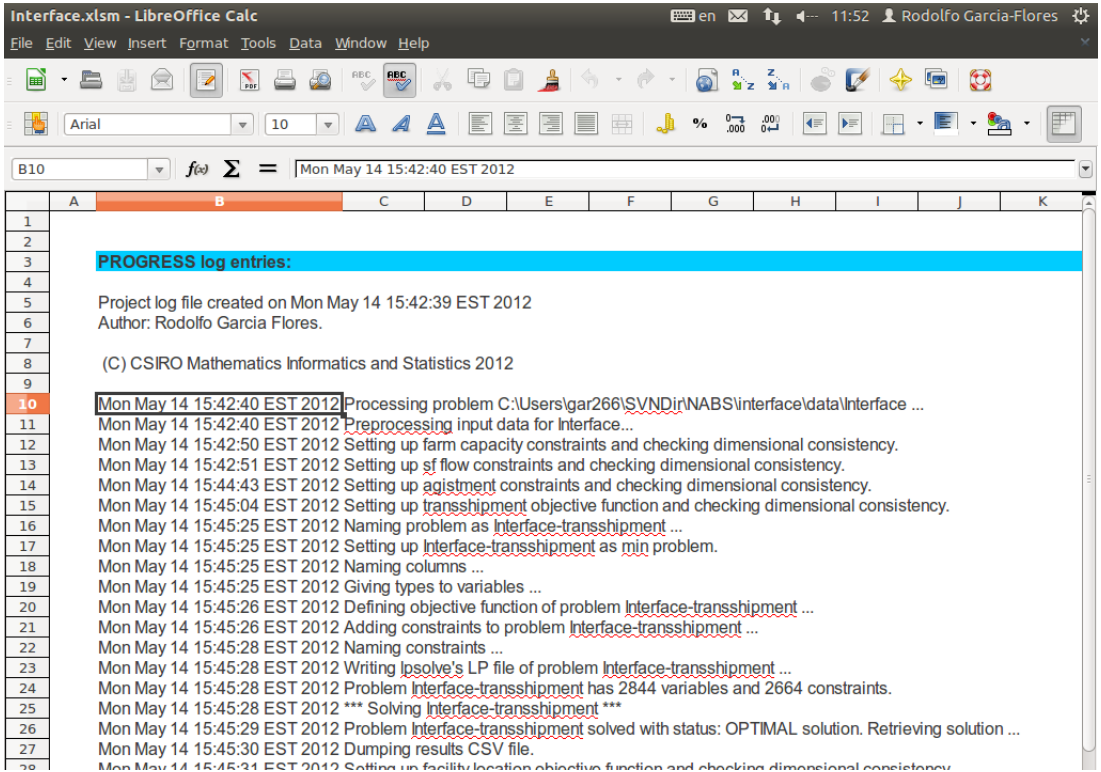
And also

12. Make it easy to reuse.
49. Test your software, or your users will.
62. Test early. Test often. Test automatically.
63. Coding ain't done until all the tests run.
66. Find bugs once.

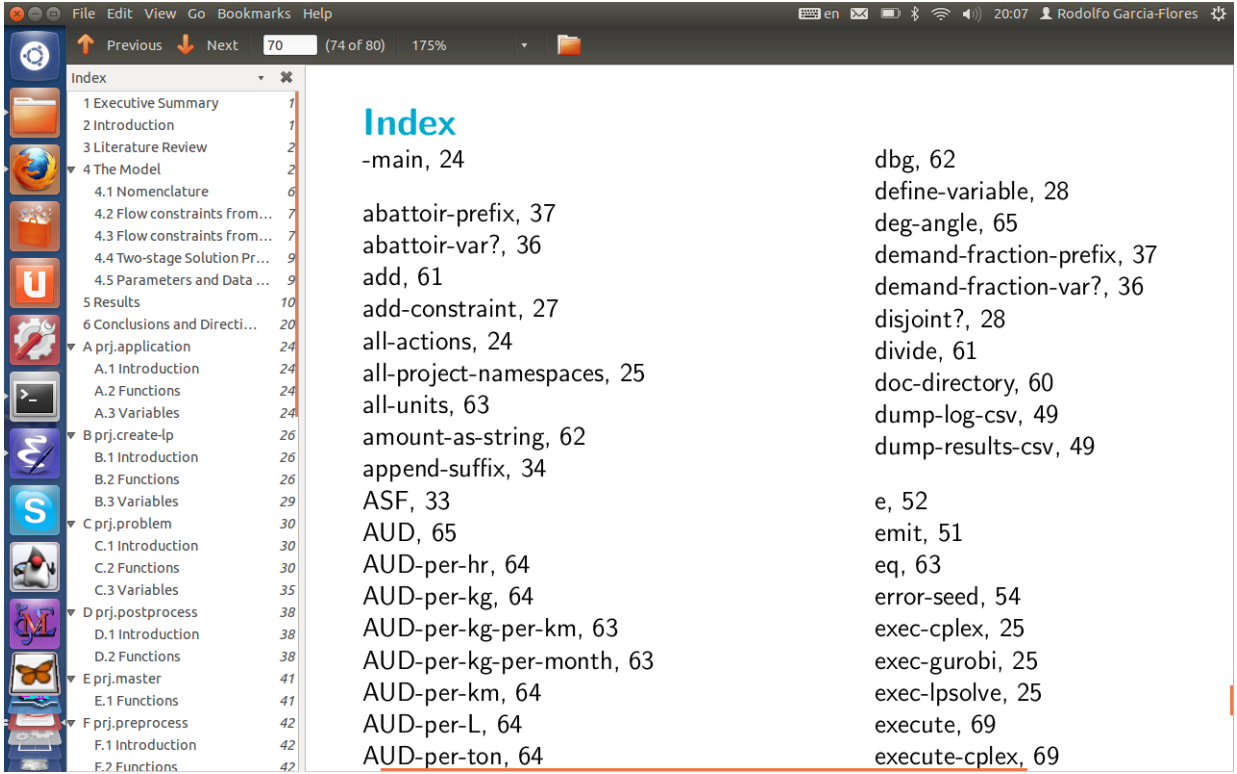
48. Design to test

```
*************************************************************
***                        TEST SUMMARY                   ***
***          (C) CSIRO 2012 - Rodolfo Garcia Flores       ***
*************************************************************
Test of module prj.application.
(C) CMIS 2009 - Rodolfo Garcia-Flores

Total # of functions: 5
Total # of lines: 179
of which:
  Test Lines: 15 (8.38%)
  Code Lines: 150 (83.8%)
  Comment Lines: 14 (7.82%)

*************************************************************

summon-action {:elapsed-ns 28097146, :result :ok}
-main {:elapsed-ns 4641, :result :ok}
test-whole-project {:elapsed-ns 3594, :result :no-test}
get-map-of-actions {:elapsed-ns 509325, :result :ok}
weave-whole-project {:elapsed-ns 2322, :result :no-test}

...

*************************************************************
Total # of functions: 177
Total # of lines: 5115
of which:
  Test Lines: 1045 (20.43%)
  Code Lines: 3483 (68.09%)
  Comment Lines: 587 (11.48%)
```

48. Design to test



54. Use a project glossary



29. Write code that writes code

- Input data is code, the solution is code.
- Macros for testing and debugging, e.g.,

```
;; As seen in http://stackoverflow.com/questions/
;;    2352020/debugging-in-clojure
(defmacro dbg[x] `(let [x# ~x] (println "dbg:" '~x "=" x#) x#))
```

- `partial` is handy:

```
(defvar maximum (partial select-extreme max))
(defvar minimum (partial select-extreme min))
(defvar gt   (partial compare-to #(= % +1))
(defvar lt   (partial compare-to #(= % -1))
(defvar eq   (partial compare-to #(= % +0))
(defvar gteq (partial compare-to #(or (= % +1) (= % 0)))
(defvar lteq (partial compare-to #(or (= % -1) (= % 0)))
```

. . . and so on with I/O, formulation, logs, tests, executables, etc.

Work to do

1. Develop a more realistic model.
2. Bring representation closer to problem domain, it is not close enough yet.
3. Improve GoogleEarth translation namespace.
4. Can we adjust memory size to parse solution?
5. Output needs polishing.
6. Build an app extension.

Thank You

CSIRO Mathematics, Informatics and Statistics

Rodolfo García-Flores
t +61 3 9545 8059
e Rodolfo.Garcia-Flores@csiro.au
w Mathematics, Informatics and Statistics web