[redacted]

TRANSPORT TABLEAU PROGRAM SPEC.

existing program needs refactoring!

vector: vector<int>.
pairs: vector<pair<int,int>>.
matrix: vector<vector<int>>.

Create new tableau class?

Transport Tableau

private helper methods.

Could these be public?

- findAdjacent (pair, candidates pairs) : pairs
- findCycle (pairs) : pairs (cycle)
- removePair (pair, pairs) : pairs
- solveUiVj() : void.

This function is only required because C++ lacks such facilities for deleting items from a vector.
Perhaps refactoring/creating a container class will solve this?

matrix vector vector
+ TransportTableau(cost, supplies, demands);
+ ~TransportTableau();
+ getAllocations() : matrix
+ getBasicPairs() : pairs.
+ getCost() : int
+ getDemands() : vector
+ getLinkFlowCost() : matrix
+ getSupplies() : vector
+ getUiValues() : vector
+ getVjValues() : vector
+ isOptimal() : boolean
+ nextTableau() : void
+ northWestCornerRule() : void.

Important methods

- linkFlowCost : matrix
- allocations : matrix
- supplies : vector
- demands : vector
- basicPairs pairs
- uiValues : vector
- vjValues : vector

The Balanced Transportation Problem is determined by:
- link-flow costs
- supplies
- demands.

(balanced : supply = demand).
- if not, either add fictitious supply or fictitious demand.
(We can add this functionality later.)

* Use namespace TransportationSolver.

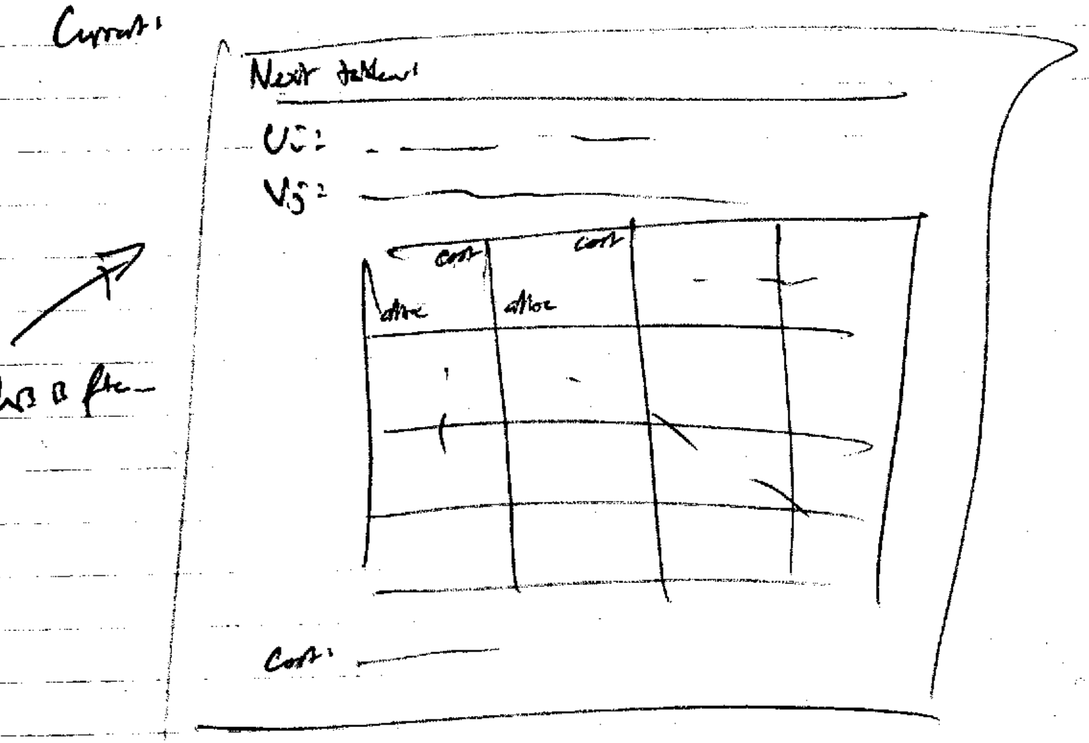
Refactoring Ideas.
- findAdjacent, findCycle, removePair are utility functions in a separate namespace.
- use std::set for pairs (we don't want duplicates!)

NEW DESIGN FOR TRANSPORTATION PROBLEM SOLVER

- The functions findAdjacent() and findCycle() appear as utility functions to the namespace transportation_solver
- If we use sets of points, we don't have to define a remove_element function.

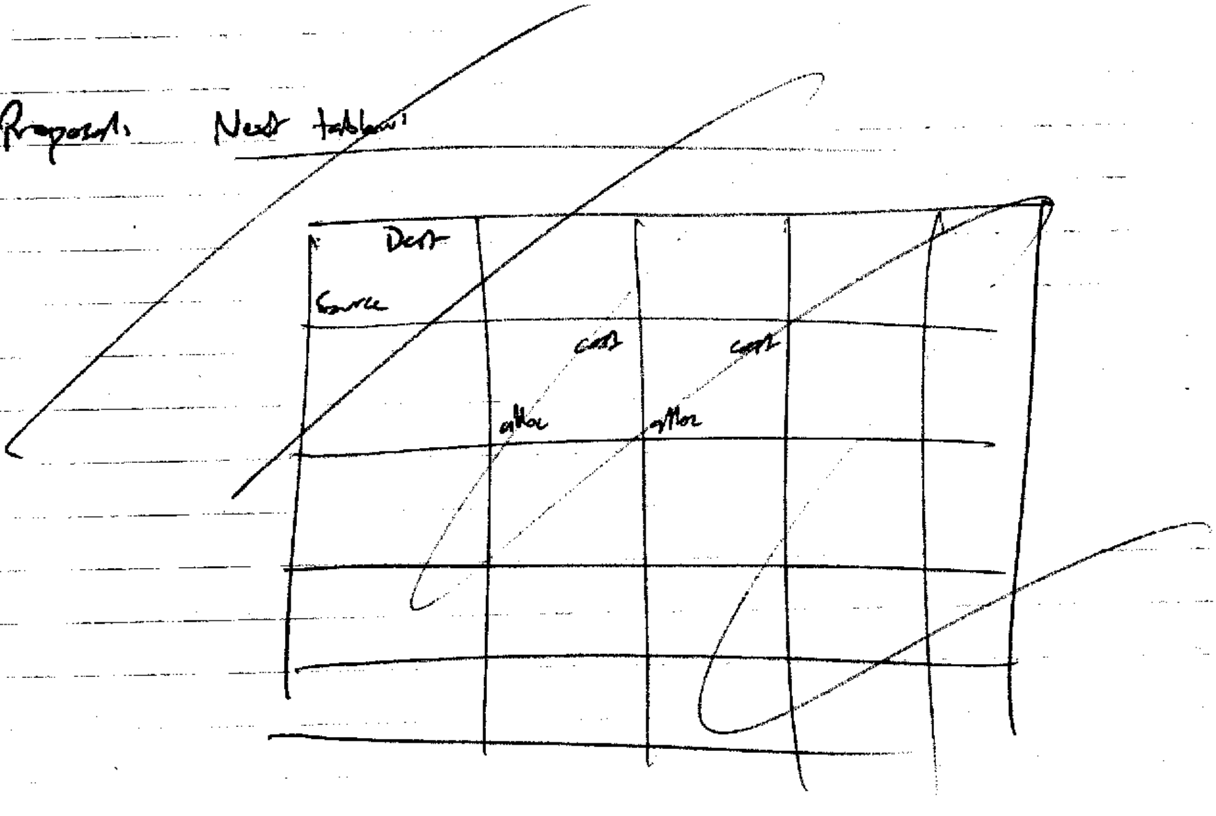
We can also change the UI while we're ~~at it:~~ N/A.

Current:



This is fine

Proposed:



Steps for making program:

files:

~~transportation_solver~~ ancillary_functions.hpp — header file for ancillary functions — find_adjacent — find_cycle. ~~findadjacent~~, ~~findcycle~~

* May want to define our own containers for ease of use here.

transportation_solver.cpp — Implementation for namespace functions

~~transportation~~ solver namespace

~~TransportTableau.cpp~~

transport_tableau.cpp
transport_tableau.hpp } TransportTableau class.

transport_tableaux namespace

* Use underscore_case for all function names

main.cpp — contains the current drawing function.

We want testing facilities for all parts of the program.

Project name: transportationSolver is taken!

Need new name: TransportTableaux — prints tableaux for a solution to the Transportation Problem in Operations Research for demonstrative purposes.

test files:

test_transport_tableau.cpp — testing for the transportation tableau class itself.
test_transportation_solver.cpp. — testing of ~~auxiliary~~ ancillary functions.

Project structure

TransportationTableaux / src / test / test_ancillary_functions.cpp
test_tableau.cpp.

↳ source files: tableau.hpp
tableau.cpp } Tableau class

ancillary_functions.hpp
ancillary_functions.cpp } Custom

✓
should this be done in C++, or another language instead?
(Java? Scala? ...)

1. define easy class for index pairs
   (set of pairs?)
2. define find-adjacent
3. find-cycle.

ANCILLARY FUNCTIONS

namespace transportation_tableaux {
    typedef PairSet = std::set< ~~std::~~pair<int,int> , comparator >;

< compare function)

If first < first true
(else if second < second true.
else false

use unordered-set.

~~Set~~

* use an unordered set of pairs.

typedef ~~IndexSet~~ std::unordered_set < std::pair<int,int> >
IndexSet

Traverse with iterator.

member functions
- insert (value)
- size()
-

1/ define an Index class:

2/ overload ==
&
!=

class ~~Index~~ IndexPair {
    ~~int i, int j;~~
    operator < (other)

typedef PairSet = std::set < std::pair<int, int>, PairComparator >;

Struct PairComparator {
    bool operator ()(const pair<...>&a, const pair<—> &b)
    {

}.

PairSet test cases:

check — instantiation works with comparator.
    Pairset sorts properly.
    Pairset maintains uniqueness of values.
    PairSet properly handles adding and removing pairs.

ANCILLARY FUNCTION TESTING

test_ancillary_functions.cpp Boost Module Test Cases.

Test: • ~~PairSet~~ IntPairSet + works with IntPairComparator.

• find_adjacent_pairs function:

IntPairSet find_adjacent_pairs(IntPair, IntPairSet)

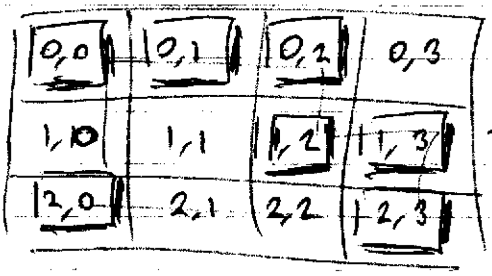
• find_cycle function

IntPairSet find_cycle(IntPair).

IntPairSet test cases are as previous.

The other functions can be ~~[illegible]~~ tested individually for some scenarios, but we can also devise grids that have a known answer and test their behaviour at the same time.

eg.



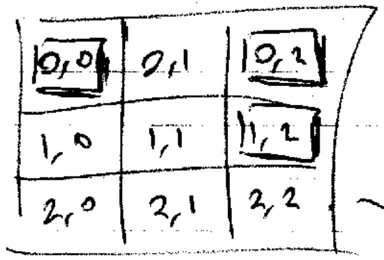
0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

eg. 2,0 adjacent to 0,0, 2,3.

cycle contains all pairs. (edge case: no pruning should happen.)

* PROBLEM: For the +, - alternation to work, should 0,1 not be included (ie. the cycle goes 0,0 → 0,2, skipping 0,1)? Will need to investigate this.

Other boundary cases:



0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

• 0,0 adjacent only to 0,2.

• no cycle (ie. pruning should eliminate all ~~[illegible]~~ indices from the set.)

Should this really be a boundary case?

7/6/2014

• New idea: have separate prune_to_cycle function.

find_cycle then calls this after removing the elements in turn to find the minimum cycle for a grid.

Fixes the above problem!

* Compile in C++11. using, not typedef!

PROJECT NAME: TRANSPORTATION TABLEAUX.

COMPILING NOTES:

- for test suite, need to compile with -lboost_unit_test_framework.
- for C++11 support, need to compile with -std=c++11.

TESTING ANCILLARY FUNCTIONS:

compile with

```
g++ -std=c++11 -lboost_unit_test_framework -o test_ancillary_functions
        src/ancillary_functions.cpp test/test_ancillary-functions.cpp.
```

Tasks:

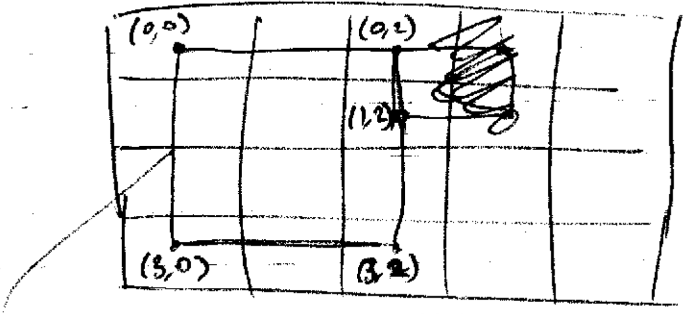
find-adjacent-pairs ||
find_cycle
remove_redundant cycle pairs

remove_redundant
_cycle_pairs.

1. finish ancillary functions definition + implementation.

   & We need to find a way to get the minimal cycle once we've pruned it
   (i.e. discard any redundant pairs from the cycle.)

eg.



Idea: we can simply remove each element one by one, prune to a cycle ∴ if the element that was removed was vital to the cycle structure, then we should get nothing.
If we still have a cycle, then the removed node was redundant.
(CAN THIS BE MORE EFFICIENT THOUGH?)

2. finish test cases (at least ~30.) The test cases file can always be modified with more tests later.

TESTING TABLEAU CLASS:

compile with

```
g++ -std=c++11 -lboost-unit_test_framework -o test_tableau
        src/ancillary-functions.cpp src/tableau.cpp test/test_tableau.cpp.
```

We need to compile ancillary_functions.cpp also, since it contains vital functions and type definitions.

TESTING TABLEAU CLASS:

Tasks: 1. finish tableau definition + implementation.

(should be easy with ~~these~~ few things kept in mind:

- compile for std=c++11, so we can use ranged-fors. Also take into account move semantics if relevant.
- tableau should use the IntPair, IntMatrix types defined in the ancillary-functions header, and use its functions.

Tableau class

unit flow costs.
+ Tableau (costMatrix, supplies, demands)
int matrix, int vector, int vector
+ ~Tableau()

+ ~~get_alloc~~ get_allocations(): matrix
+ get_basic_pairs(): IntPairSet
+ get_cost(): ~~matrix~~ int.
+ get_demands(): vector<int>
+ get_unit_flow_costs(): matrix
+ get_supplies(): vector<int>
+ get_ui_values(): vector<int>
+ get_vj_values(): vector<int>
+ is_optimal(): bool
+ next_tableau(): void
+ northwest_corner_rule(): void
+ solve_ui_vj(): void

4 - ~~unitflow~~ unit_flow_costs : matrix
1 - allocations : matrix
5 - supplies : vector<int>
3 - demands : vector<int>
2 - basic_pairs : IntPairSet
6 - ui_values : vector<int>
7 - vj_values : vector<int>.

alphabetical order

* Also store + retrieve star pair + pairs - pairs.

use find_adjacent, prune_to_cycle, find_cycle from ancillary_functions. RemovePair is no longer needed now that we're using a set data structure.

* NO PRIVATE METHODS!

+ get_traversal_order_cycle()
+ get_star_pair();
~~all~~ ~~solve~~

this would need to be a vector...

Store current ~~cycle~~ traversal order cycle locally? (this would allow it to be displayed in the output.) cycle_traversal.

IntPairSet starting from * pair.

+
- star_pair : IntPair.

2. finish test cases (~40.) We can add more tests as needed.

TRANSPORTATION TABLEAUX.

MAIN PROGRAM (main.cpp)

1. Should be able to copy over from existing Driver with minor modifications. (This will be the first draft of the program. Might look into a GUI implementation later.)

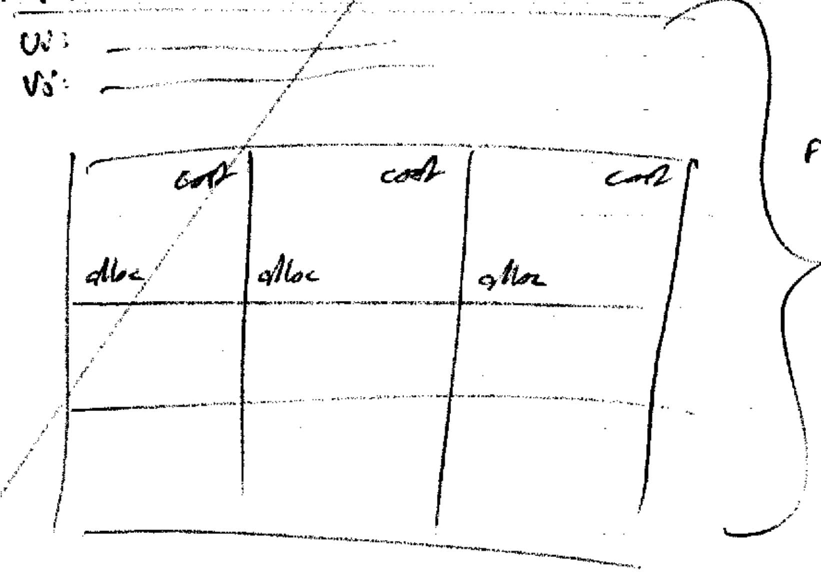
* 13/6/2014: Problem:

The traversal order for the cycle should be stored in a vector (order is important), not a set. This added an additional type definition:

- The Tableau object is quite large and cumbersome with the addition of tracking the star-pair, cycle-traversal-order and +,- values at each point. Should it be refactored?
  - Should be fine for now. The interface stays the same, so we'll try this way and see if it is efficient or not.

Update the UI?

Current Next tableau:



refactor with a cycle class that contains: star pair, traversal order

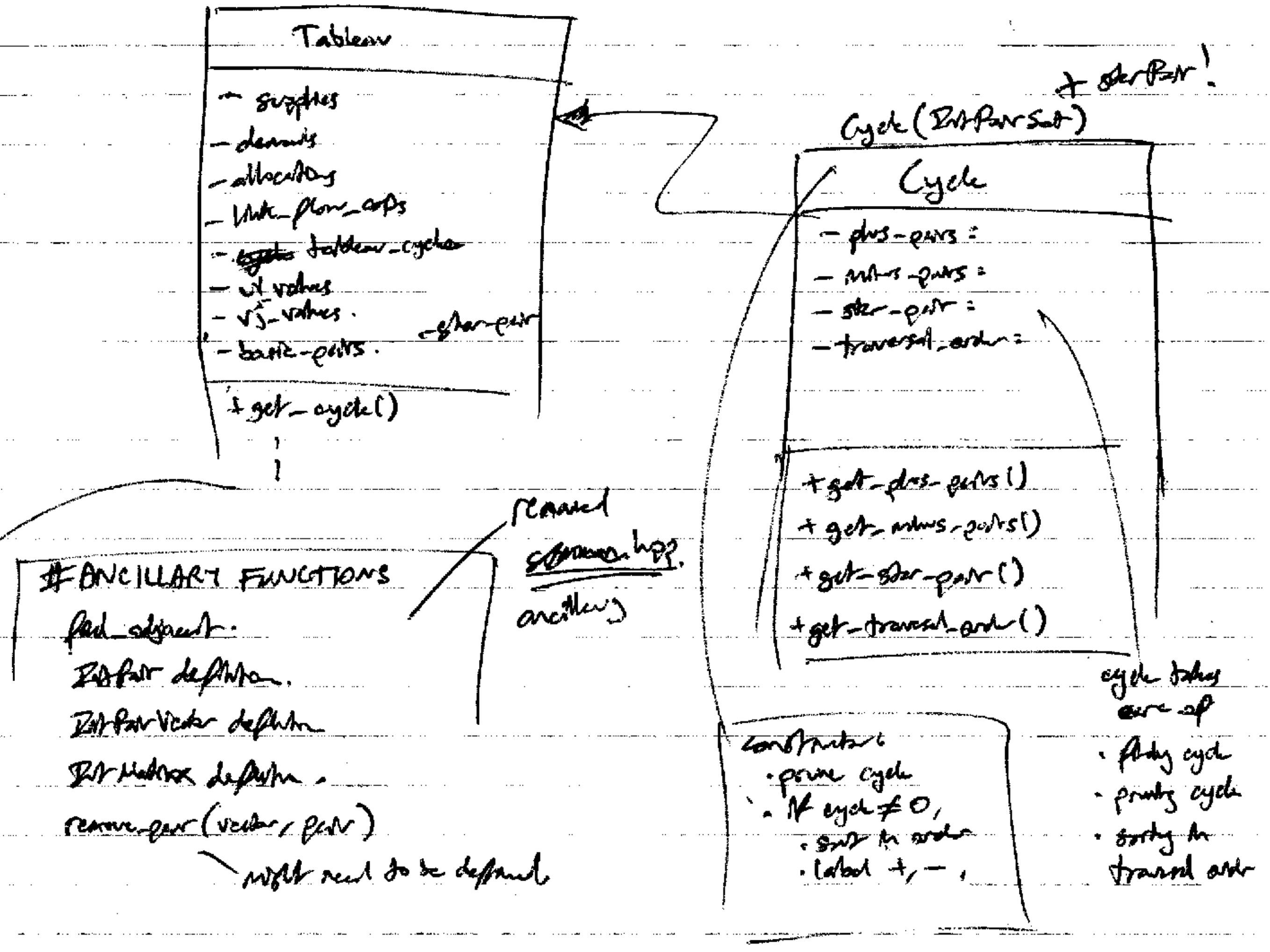
* If we keep track of the star pair, we can modularize the main algorithm.

```
+ next_tableau() ——> + find_star_pair();                 ~~cycle~~ tableau
                     + find_cycle();                     cycle
                     + remove_cycle_redundant_pairs();   cycle
                     + find_traversal_order_cycle();     cycle
                     + label_plus_minus();               cycle
  adjust_costs() ——> + update_basic_pairs();             tableau
                     + solve_ui_vj'();                   Tableau
```

(should this even be a method!)

TRANSPORTATION TABLEAU

New program design:



| normal accessor methods
northwest_corner_rule()
B_optimal()
solve_ui_vj()
find_cycle()
find_star_pair()
~~adjust_basic_pairs~~()
adjust_cycle_allocations()
update_base_pairs()

YAGNI:

leave the star_pair / get cycle stuff until after the main program works!

14/6

* having find_cycle here allows us to test it!

REFACTORING:

3 COMPONENTS:

ALL UNDER transportation_tableaux namespace

ANCILLARY

* [illegible] printing methods!

Functions:

- find_adjacent_pairs (pair, pairs) : pairs
- find_cycle (star pair, candidates) : cyclic IntPairVector — otherwise circular dependencies!

  proceeds to cycle, removes redundant pairs, returns a cycle object.
- sum_vector (vector)
- remove_pair (from vector)
- balance_supply_demand (supply, demand)

TYPEDEFS:

- IntPair
- IntPairVector
- IntMatrix

CYCLE TRAVERSAL — new name.

* cycle shouldn't be responsible for printing itself! (the function should be in the Ancillary container.)

Does:

- In the constructor:
- determine traversal order — if not possible, error.
- determine + pairs
- determine − pairs.

Keeps track of:

- plus pairs
- minus pairs
- traversal order
- star pair.

TABLEAU

Keeps track of:

- unit_flow_costs
- basic_pairs
- supplies
- demands
- ~~tableau cycle~~ cycle traversal (call cycle).
- ui_values
- vj_values
- allocations.

Does:

- In the constructor, simply set up values.

Algorithm stuff:

1. find star pair
2. find cycle
3. adjust ~~[illegible]~~ alloc.
4. update basic pairs
5. solve ui_vj.

Northwest corner rule.

Program steps:

~~1. MAIN — prompt/get unit flow costs, supplies, demands. Set up Tableau. (Print initial tableau?)~~

~~2. MAIN — call tableau.northwest_corner_rule(). Print tableau.~~

```
1. MAIN — prompt/get unit flow costs, supplies, demands.
2. MAIN — If not balanced, add fictitious supply/demand. Print helpful message.   — call ANCILLARY METHODS.
3. MAIN — construct the Tableau. Print this initial tableau.
4. MAIN — call tableau.northwest_corner_rule(). Print tableau.
5. MAIN  [ WHILE (tableau.is_optimal() == false)
            [ 6.  TABLEAU — find star pair()
            | 7.  ~~TABLEAU~~ ANCILLARY — find_cycle (basic pairs, star pair)
adjust_allocations() — 8.  TABLEAU — construct cycle traversal (pairs).
            | 9.  TABLEAU — adjust_cycle ~~cost~~ allocation ( )
            [ 10. TABLEAU — update_basic_pairs ()
            | 11. TABLEAU — solve_ui_vj ()
            [ 12. MAIN — print new tableau.

13. END PROGRAM.
```

REFACTORING

Class UML Diagrams.

ADD:
adjust_~~[illegible]~~_allocations()
determine_star_pair()
determine_cycle()

TABLEAU

Tableau (unit_flow, supply, demand)
~Tableau()
~~+ adjust_allocations() : void~~
+ ~~find_star_pair() : IntPair~~
+ get_allocations () : IntMatrix
+ get_basic_pairs() : IntPairVector
+ get_cost() : int
+ get_demands() : ~~IntP~~ vector<int>
+ get_unit_flow_costs() : IntMatrix
+ get_supplies () : vector<int>
+ get_ui_values () : vector<int>
+ get_vj_values () : vector<int>
+ is_optimal () : bool
+ northwest_corner_rule () : void
+ solve_ui_vj () : void

Program component interface.

- allocations : IntMatrix
- basic_pairs : IntPairVector
~~demands~~
- cycle : CycleTraversal
- demands : vector<int>
- unit_flow_costs : IntMatrix
- supplies : vector<int>
- ui_values : vector<int>
- vj_values : vector<int>

we don't store the star pair locally.
(except in the cycle traversal object).

ADJUST_ALLOCATIONS() function
1. find star_pair
2. ANCILLARY — find cycle
3. construct cycle traversal / store cycle.
4. adjust allocations.
5 update basic pairs.
└ doesn't need its own method — last line.

+ ~~get_cycle_traversal~~
+ get_cycle ()

We need a find_cycle() method —

CYCLETRAVERSAL

Cycle (~~cycle pairs~~ star pair, cycle pairs)
~Cycle()
+begin() +end()
+ get_minus_pairs() : pairs
+ get_plus_pairs() : pairs
+ get_star_pair() : pair
+ ~~get_traversal_order~~():

OR: Iterators?
begin(), end() ?

Try this — we can use ranged-for loops!

- traversal_order : IntPairVector
- star_pair : IntPair
- minus_pairs : IntPairVector
- plus_pairs : IntPairVector.

// ANCILLARY HEADER //

Typedefs: IntPair, IntPairVector, IntMatrix
— no IntPair vector defs!

Functions:
balance ~~[illegible]~~ problem (supply, demand, unit_flow_costs) : void
find_adjacent_pairs (pair, pairs) : pairs
find_cycle (star_pair, pairs) : pairs
remove_pair (pair, vector) : void
sum_~~vector~~ elements (vector) : int

~~Instead~~: include <numeric> accumulate(...).

also need to add 0 column/row to unit flow matrix!

should this be separated into multiple functions?

15/6

* Idea! Also do tutorial-style user documentation!

ANCILLARY FUNCTIONS

Simple functions:

* sum_elements: add up all integer elements in vector.
We can use the STL accumulate method to do this:

```
std::accumulate(elements.begin(), elements.end(), 0);
```

→ this value is returned.

i.e. $\underbrace{0 + \text{element values}}_{\text{accumulating.}}$

Test cases:

Element Sum.

1. test that it sums up correctly for positive values
2. " " negative values
3. test that it sums up duplicates / 0 elements — eg. {0, 0, 0, ..., 0}.
4. test that it sums up an empty vector — no elements.
5. test mixed elements
6. test idempotency. (no side-effect output.)

Test with

```
g++ -std=c++11 -lboost_unit_test_framework -o test_ancillary
    ../src/ancillary.cpp test_ancillary.cpp.
```

* Remove pair: given a pair and a vector, remove the pair if it exists.
  - if pair does not exist, do nothing.
  - if pair exists multiple times, only remove the first instance.
  - can use the find function from the <algorithm> library.

Test cases:

1. test that it removes an existing pair correctly
2. test that it does nothing when trying to remove a non-existent pair
3. test that it only removes the first instance of a duplicate.

Test with

```
g++ -std=c++11 -lboost_unit_test_framework -o test_ancillary
    ../src/ancillary.cpp test_ancillary.cpp.
```

eventually we'll test with -Wall.

ANCILLARY FUNCTIONS

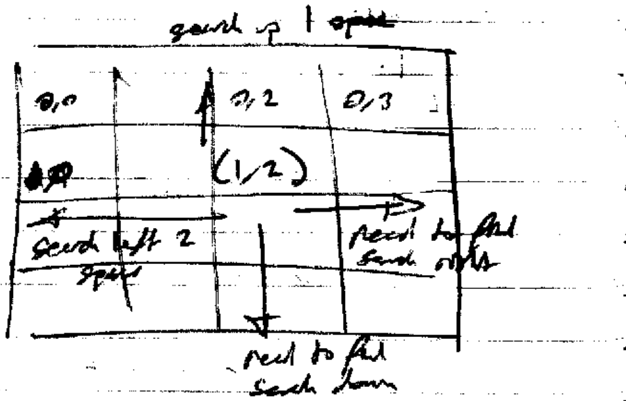
Complex ancillary functions: find_adjacent_pairs (pair, candidates)
find_cycle (pair*, basic_pairs)

find_adjacent_pairs (pair, candidates)

Algorithm

for pair (1,2)

1. find the size of the search grid.
   (for pairs (x,y) in the candidate pair set, the size of the grid is the max of the x and y values together.
   i.e. height = x, width = y.
   rows columns
2. Search up, add first adjacent if found
3. Search down, add 1st adjacent if found.
4. Search left, add first adjacent if found.
5. Search right, add first adjacent if found.
6. Return adjacent.



for general (x,y)
- search left 0+y spaces >0
- search right width − y spaces <width
- search up 0+x spaces >0
- search down height−x spaces <height

Can this algorithm be optimised by only considering pairs with the same x,y value?

let (x,y) denote the

```
1. for each pair in candidates: (p,q)
     ~~if (p=x) ∨ (q=y) {~~
     if (p=x) {
         if (q<y) LEFT;
         if (q>y) RIGHT;
     }
     if (q=y) {
         if (p<x) UP;
         if (p>x) DOWN;
     }
```

We construct lists containing those pairs that are left, right, up or down from the given pair.

2. if any list has size > 1, find closest pair. Else simply add to adjacent.
3. Return adjacent.

(x,y)
left: closest is largest y.
right: closest is smallest y.
up: closest is largest x.
right: closest is smallest x.

if left, up largest
if right, down smallest

Seems more efficient.
Try this method first.

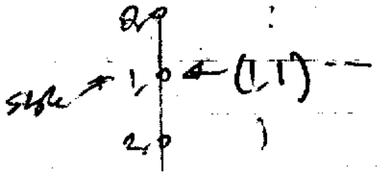
ANCILLARY FUNCTIONS

find_adjacent_pairs (pair, candidates)

there might be some trouble with the lambda and referencing in conjunction with the STL classes.
If errors occur, check those first.

Test cases:

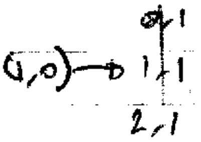
✓2 1. Test that the left works eg.



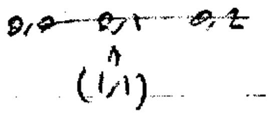
also do multiple.
(1,0) (1,1) (1,2) ← (1,3)

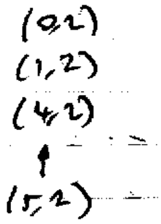
other directions should be empty!

✓2 2. Test that the right works eg.
(1,1)
(2,0) → (2,1) (2,2)(2,7)

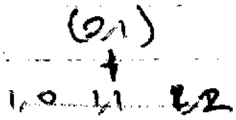


✓2 3. Test that the up direction works eg.
also multiple

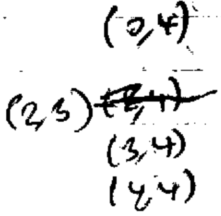




✓2 4. Test that the down direction works, eg

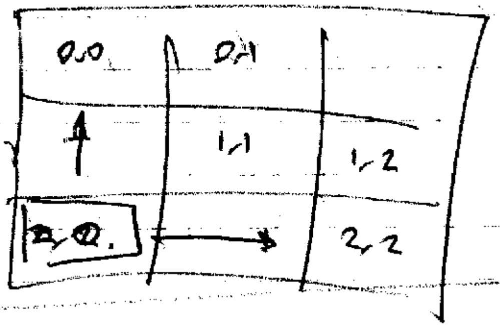


5, 6, 7, 8 are multi element versions.



9. Test that it works on a general case.

Case 01



* Also need to have an idempotency test at some stage.

15/6 Still need to implement the find_cycle() method, but will work on finishing the cycle_traversal class stuff for now.

- CycleTraversal class fully documented: just need to implement constructor.
  Also need to write unit tests.

* The tableau class needs the find_cycle() method!

```
Program { nw_corner_rule
          while (!optimal) {
          ~~[illegible]~~ [illegible]_path()
          ~~find_cycle~~)
          determine_cycle();
          adjust_cycle_allocations();
```

15/6/16/6

constructor DOCUMENTATION

Tasks:
- finish Tableau.hpp documentation ✓
- Tableau.cpp — northwest_corner_rule() ✓
- solve_ui_vj() ✓

Should Tableau have a next_tableau() method? ✓

- Ancillary header — find_cycle() method.
- CycleTraversal — constructor implementation

program components. Come up with efficient algorithms for these.

Then testing everything:
1. Test cases for CycleTraversal
2. Test cases for Tableau.

Then work on main.cpp Driver for the program

prompt for Problem spec + balance if necessary.

Algorithm:

```
northwest_corner_rule()
while (tableau.is_optimal() = false) {
    determine_star_pair();
    determine_cycle();
    adjust_allocations();
    solve_ui_vj();
        PRINT TABLEAU HERE
}
```

* Also: 1. set up github account + project TransportationTableaux
2. print Doxygen documentation reference
3. create tutorial-style user guide.

see if we can make some functions const? (const-correctness)

Also CycleTraversal has its member's declared const.

. Also Tableau.

* For tableau, need to use a pointer to a cycle —

unique_ptr?

~~* Need a copy constructor for CycleTraversal.~~

* In tableau.cpp, replace star_pair=NULL with star_pair.reset();

!

Check that this all works correctly. (esp. get_cycle() and ~~get_star_pair~~)

17/6 Working on TRANSPORTATION TABLEAUX Project.

Tasks still to do

- tableau.cpp — check everything works correctly.
  otherwise done (code compiles without errors)

COMPONENT CODING

- tableau.hpp — finish documentation
  - constructor
  - class usage example
  } + check everything.
- ✓ cycle_traversal.cpp — finish constructor implementation
  (make sure it works with const objects!)
- ✓ cycle_traversal.hpp — finish documentation
  - class usage example.
  } + check everything.
- ✓ ancillary.cpp — implement the find-cycle method
  (prune to cycle + check for redundant elements)
- ✓ ancillary.hpp — done for now (check everything).

COMPONENT TESTING

- ✓ test_ancillary.cpp — update to test find_cycle()
  - add more test cases. — ✓ Non-doxygen comments on the test cases!
- ✓ test_cycle_traversal.cpp — start + write test cases for class.
- test_tableau.cpp — start + write test cases for class.

MAIN PROGRAM

call ~~transportation_tableaux.cpp~~ main.cpp
~~conflicts with [illegible] [illegible] [illegible].cpp~~

- ~~main.~~cpp — write the caller program code.
  (should be mostly the same as the existing code). ✓
  - check with some examples.)
- ✓ write Makefile for project. ✓ — changed it to [illegible]

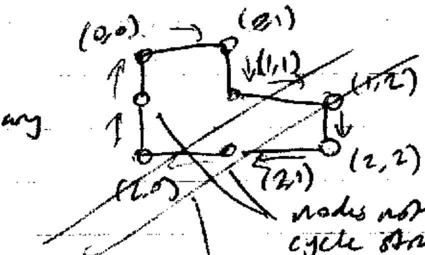
DOCUMENTATION + PUBLISHING

- print API reference for components using Doxygen.
- create Tutorial-style guide in Latex (with pictures).
- ✓ create ~~[illegible]~~ README.md.
- ✓ finalize [illegible] commit of project on GitHub.
- ✓ need LICENSE file and .gitignore file.

23/6

. finishing ancillary header + implementation.
- check header
- implement find-cycle() method.
- check implementation
- test everything.

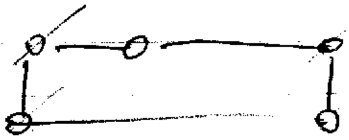
find_cycle() method: 1. prune to cycle
2. eliminate unnecessary nodes.



nodes not necessary for cycle structure.

∴ We can always get rid of the unnecessary nodes by systematically checking whether the removal of each one compromises the structure of the cycle.

Case:



We aren't going to ever get a case like this, so it should be fine.

This algorithm seems inefficient: can we do better?

```
find-cycle()
    1. Define auto prune_to-cycle()          auto prune-to-cycle =
                                             [
    2. for cycle parameter, prune_to_cycle().
    3. for each path in the cycle, remove path + prune_to_cycle12
       If return value ≠ empty,
           newCycle = return value.
       end
    4. Return new cycles
```

pure function — doesn't change when applied to the same input

Idempotency — $f(f(x)) = f(x)$.

Test cases:
- test whether it returns nothing when no cycle exists.
- test idempotency.

find-adjacent paths() idempotence → (1,0) (1,1) (1,2) (1,3)
(2,1)

purity.

(0,0)
(1,1) (1,2)
(3,1) (1,3)

find_adjacent()
no adjacent

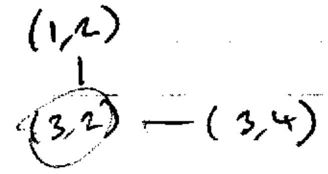
(1,1)
(2,2)
(3,1)
(4,1)
(4,4)

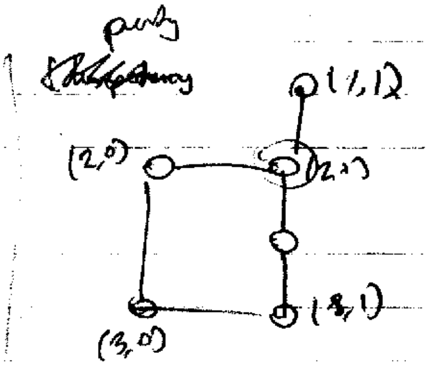
TESTING ANCILLARY FUNCTIONS:

find_adjacent_pairs() —— pure, idempotent
find_cycle() —— pure, ~~idempotent~~ not idempotent: it adds the start coor.
remove_pair() —— pure? ———— perhaps not, it acts on the input vector directly.
sum_elements() —— pure.

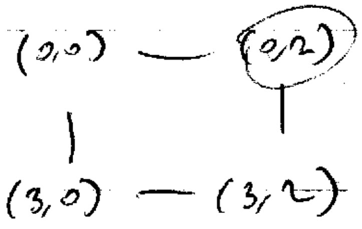
find_cycle() test cases: ———— also add to find_adjacent_pairs()? also, test for no adjacent.

1. empty input
2. no cycle

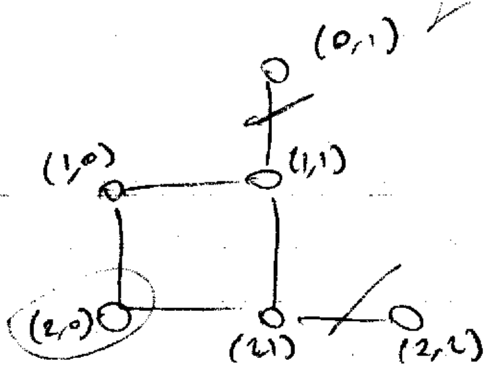




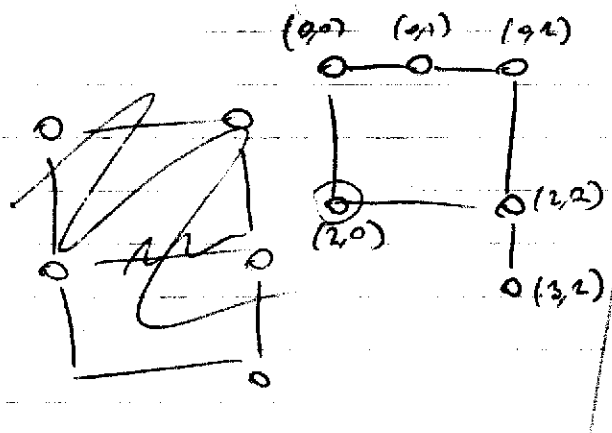
3. only_cycle



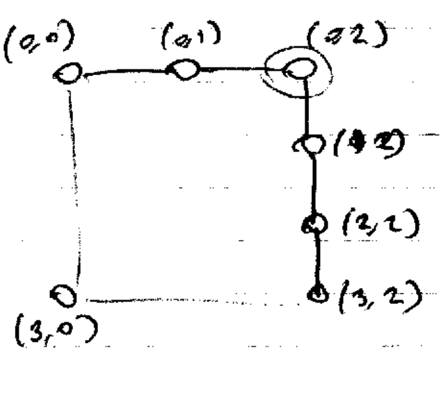
4. no redundant_pairs



5. one redundant_pair



6. multiple_redundant_pairs.



23/6.

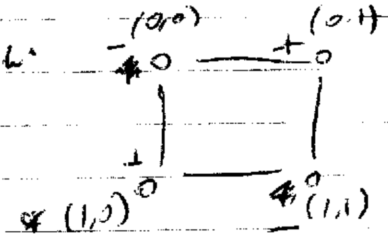
Tested the ancillary code — no errors.

Will probably want more test cases, but for now we can safely assume that it works.

24/6. · Finish the cycle_traversal code + tests.

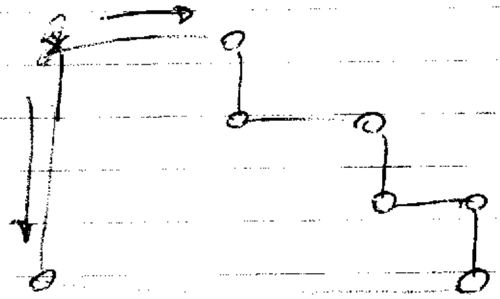
cycle-traversal constructor: given start-pair, minimal cycle pairs.

constructor example: start_pair (1,0), cycle:



sort in traversal order (1st = start_pair)

· edit. + add —



- finished cycle-traversal constructor implementation. (does work with const!)
- finished cycle-traversal header with class usage example. — class seems to work!

Test cases:

constructor — test empty cycle pairs. *

— test no cycle path.

— test a true cycle path.

What should the ~~class do~~ if

· no pairs are provided?

· no cycle can be constructed?

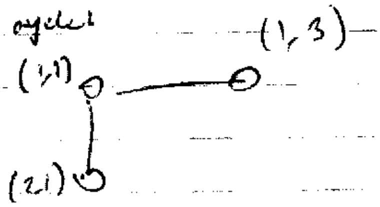
~~fail with exception.~~

return start_pair only

* For for-loops, use unsigned int instead of int?

(gets rid of the compiler warnings.

No cycle



// Refactored with a ~~method~~ function that automatically compares all cycle-traversal details with an input parameter list.

* ~~[illegible]~~ Test that the class can even hold the [illegible]!

Need to add more test cases for cycle-traversal!

29/6 — copied over code for main.cpp and fixed some bugs in the solve_using method.
Program works as well as its predecessor version at this point.

Still to do:

1 · tableau.hpp — finish documentation + class usage example.
2 · test_tableau.cpp — make test cases
✓3 - upload to github.
30/6 {
✓4. add LICENSE file
✓5- add gitignore file.
✓6. add Readme file.

* Modifying existing find_cycle function defn/header (it doesn't need the start point!)

ST~~ILL NEED TO DO~~ THIS ✓ DONE

0 {
- modify header interface ✓
- modify code in cpp implementation ✓
- modify all find_cycle test code. ✓

Still to do:

✓1 - finish tableau.hpp documentation + class usage ✓
2 - test files! (need to make cases for tableau.cpp.)
3 - fix up main (need to test for balanced problem + correct if necessary.)
✓4. print API Doxygen reference.
5. use LaTeX to create a tutorial style user guide.
~~6 Add to~~ .gitignore file.

4/7

Tableau.hpp finalisation.

Class usage:
Set up tableau with sup, demands, alloc.

Constructor:
Tableau(unit_flow, sup, dem)
~~also~~ get_allocations()
get_sh_cell() ~~get_shadow_~~pack() get_basic_pairs()
get_vi_values() get_cost() get_cycle()
get_vj_values() get_demands() get_supplies()
get_unit_flow_costs()
is_optimal()
next_tableau
northwest_corner_rule()

need to include <algorithm> in tableau.hpp.

* Modification:
In tableau.cpp, use std::min_element() with a defined comparator function. in the adjust_allocations() method.

```
auto comp_allocations = [] (pair1, pair2) {
    return alloc[pair1] < alloc[pair2]
}
```

10/7 TRANSPORTATION TABLEAUX UPDATE:

1. Modifying Tableau::adjust_allocations() to use the <algorithm> libraries' std::min_element() function with a lambda comparator:

Problem: Need to find the pair in minus_pairs with the minimum allocation[r][c].

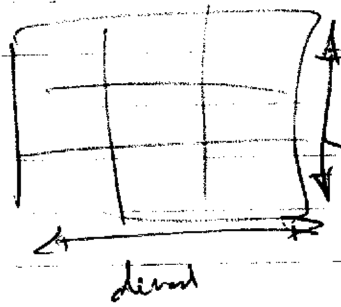
✓

Comparator lambda:

this . [&] (const pair& p1, const pair& p2) -> bool {
~~&allocations~~ & return (allocations[p1.first][p1.second] < allocations[p2.first][p2.second]);
}

2. Also adjusting main() to account for unbalanced supply-demand:

* Need to go back and clean up (update Headings) files after main today is finished.



add column of 0s for supply.

add row of 0's for demand.

Main things left to do
- Write testcases for the Tableau class (test methods individually, + worked through examples.)
- LaTeX — type up User-Manual for console-version of program.

TO MAKE MODIFICATIONS.
- In tableau.cpp
  - remove unneeded commented out code
  - change int i = pair.first, j = pair.second in both blocks to

    int i = pair.first;
    int j = pair.second; } separate out for readability!

  .hpp — change next_tableau() header demonstrative ⇒ demonstration.

  - check determine_cycle for memory leak? NVM, deletes itself

Start work on GUI.

REFACTOR IN SCALA

31 7

TRANSPORTATION TABLEAUX — UPDATED SPEC.

Requirements specification:

We require a program that will allow the (possibly tech-unsavvy) user to step through the solution of a transportation problem in Integer Linear Programming, Operations Research. The program will have a GUI that is user-friendly, and simple without any unneeded extras. Since ~~immediately~~ immediate runability and cross-platform compatibility is required, it has been decided that the program will be built on the JVM using Scala and the swing GUI framework — this will also allow for fluency and mastery in the Scala programming language to be developed.

The code will be tested extensively using the ScalaTest facilities, and all unit tests will be written before component programming takes place.

The code will be stored on GitHub, and also stored locally (backup) for version control. (we will in fact override the existing project of the same name developed earlier, which will instead be stored in a tar.gz file in the source directory. (old.tar.gz)

* Focusing on correctness > efficiency.

GUI user view:

1. need to first specify problem size (supply, demand).
2. with the supply, demand specified, we require a simple way for the user to enter values for the supply, demand and unit-flow costs. This should be done immediately in the tableau that will house the resulting solution.

alternatively, a 'Go to solution' button too.

3. This marks the end of the user input. From then, the user simply pushes a 'Step' button to step through the solution:

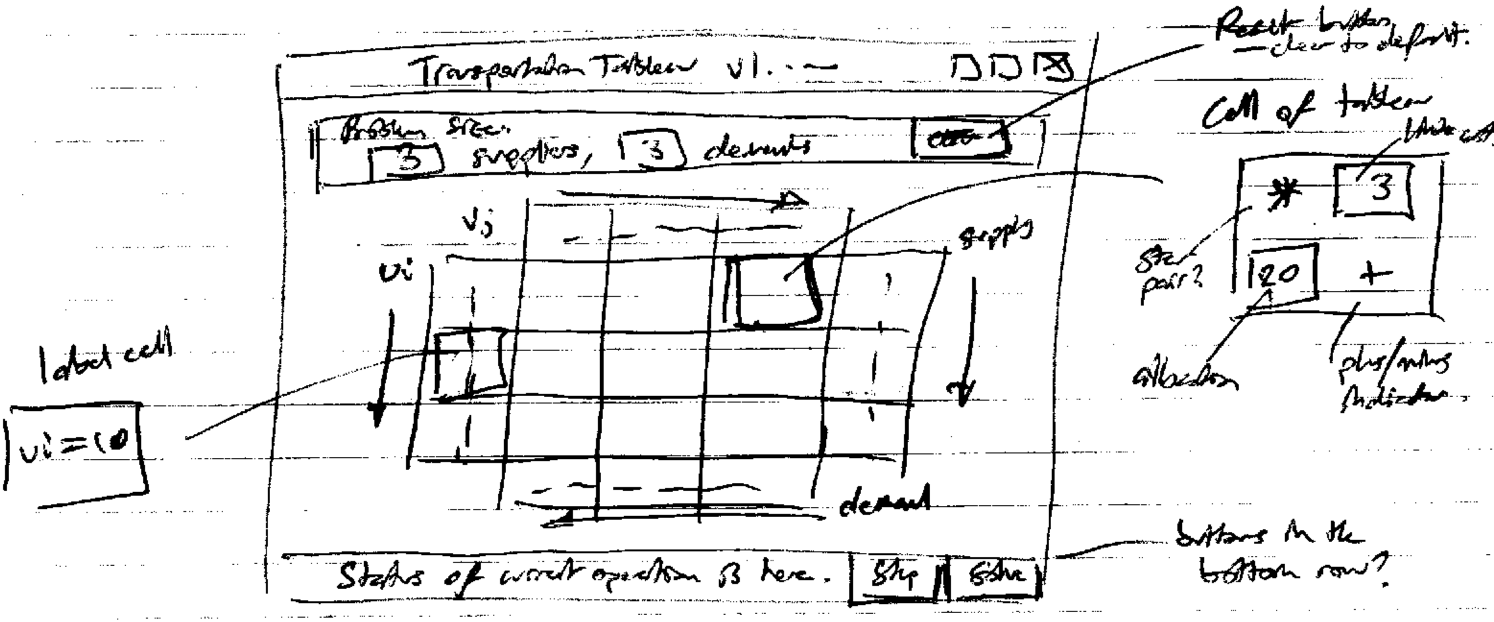
   step.
   - find the star * pair — this pair is highlighted in the table.
   - construct the tableau cycle — this is drawn to the screen.
   - adjust allocations, remove drawn cycle.
   - find u_i, v_j values
   - check for optimal solution

4. When the optimal solution is found, the user is informed.

We probably want a status that indicates each of these steps to the user as they are happening. This will not need to be multithreaded, as the operations are almost instantaneous anyway.

TRANSPORTATION TABLEAUX — UPDATED SPEC.

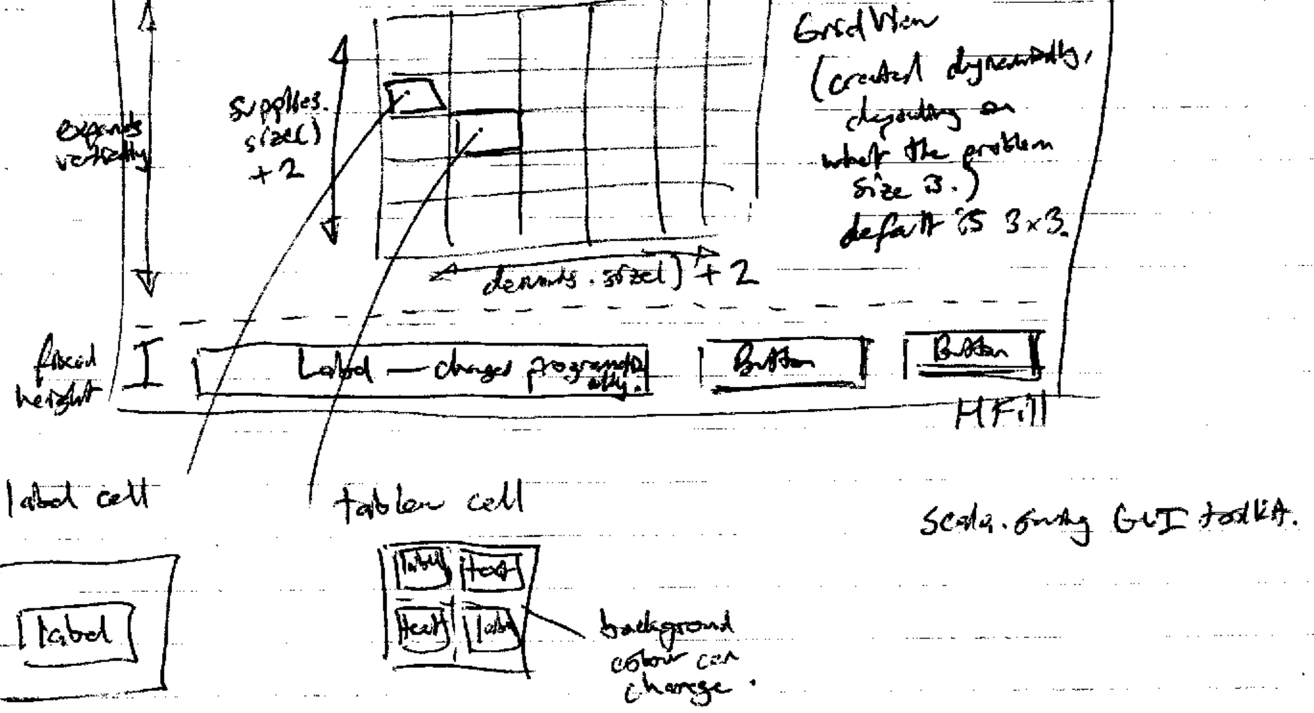
GUI user view:



(eventually the cycle will be drawn over the tableau, but that can happen last — it's not especially vital to the program operation. In the meantime we can simply change the background colours of the cycle cells or something.)

* For GUI, response time should be instantaneous.

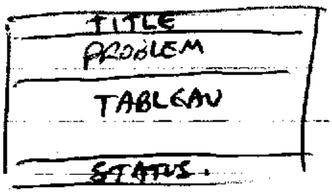
GUI — development view:



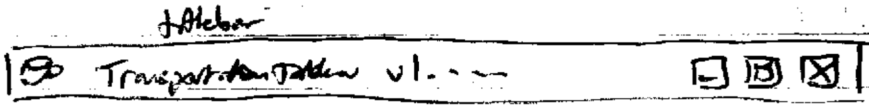
Scala.swing GUI toolkit.

TRANSPORTATION TABLEAUX — SPEC

GUI development view:



TITLE:

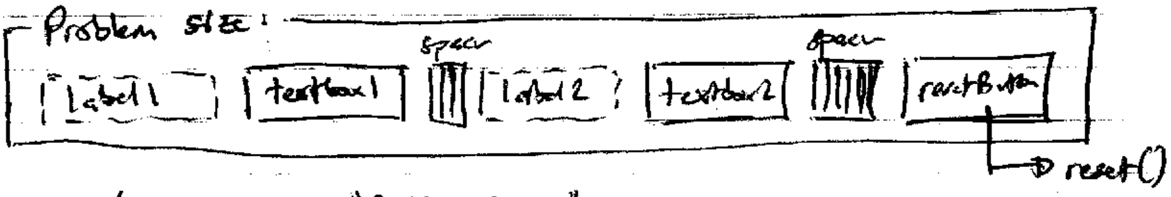


titlebar.title = "Transportation Tableau" + "v." + versionNumber

can minimise, can maximise. Window should scale appropriately.

can exit.

PROBLEM:



group w/ border title = "Problem Size:"

label1.text = "Suppliers: " textbox1.defaultText = "3", store in suppliers

label2.text = "Demands: " textbox2.defaultText = "3", store in demands

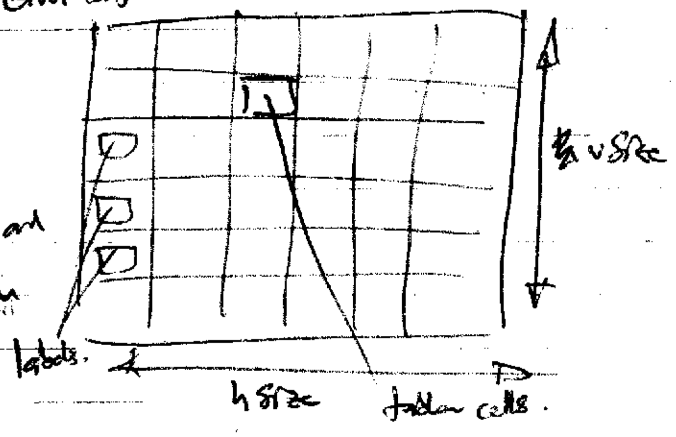
resetButton.click → reset()

(resets textbox1, textbox2 to default, resizes grid + clears any entered values. Also deletes tableau state.)

Grid layout with visible cell boundaries should work.

TABLEAU:

* In case of errors, ignore input, blank out step and solve buttons, error message in status

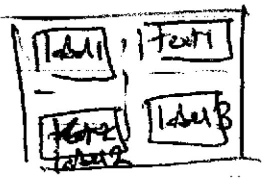


vSize = suppliers

hSize = demands

tableau cell (separate widget)

methods:
setTransfer pair ()
set cost ()
set allocation ()
set plus minus ()



label1.text = "*" if star pair, " " else.

label3.text = "+" if plus pair,
"–" if minus pair,
" " else.

label2.text = alloc[i][j].

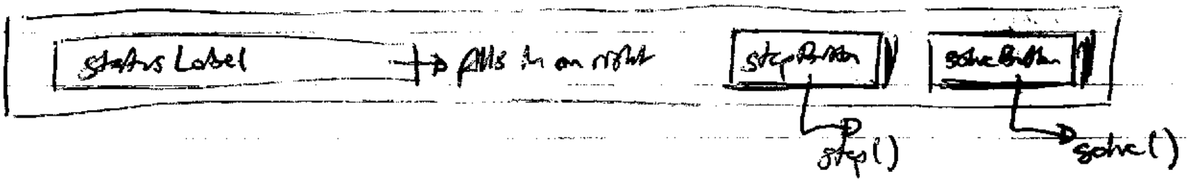
text1.text, store in cost[i][j].

31 7

TRANSPORTATION TABLEAUX — SPEC.

GUI development view:

STATUS:



Program statuses: ✓ hardcoded strings (not too worried about i18n.)

INTRO MESSAGES:

"Welcome! Enter the number of supplies and demands for the problem".

"Enter the supplies, demands and unit-flow costs into the tableau."

"Press 'Step' to step through the optimisation, or 'Solve' to jump to the optimal solution".

(should this be static at the top?)

ERROR MESSAGES:

"Invalid number of supplies / demands."

"Invalid supply/demand / unit-flow cost."

OPERATIONAL MESSAGES:

"Applying NW corner rule."

1 "Determined star pair (i,j)."

2 "Found allocation adjustment cycle."

3 "Adjusted allocations."

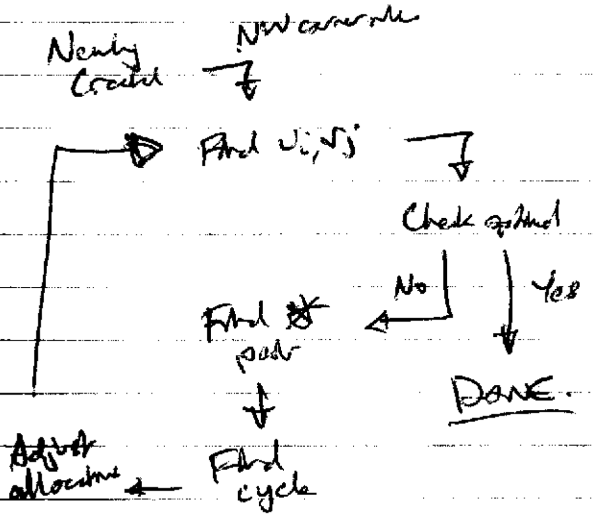
4 "Finding ui, vj for dual feasibility."

5 "Optimal solution found."

OR "Solution not optimal. Click 'Step' to continue the optimisation process."

solveButton.click → solve() (Immediately jumps to solution.), + BLANK buttons.

stepButton.click → step() . Depending on where the program is up to.



if newly created → ~~check optimal~~ find ui, vj → check optimal

~~if not opt~~

if optimal → BLANK buttons.

if not optimal →

if no star pair → find star pair

if star pair → find cycle.

if cycle → adjust allocations

if alloc. adjusted → find ui, vj

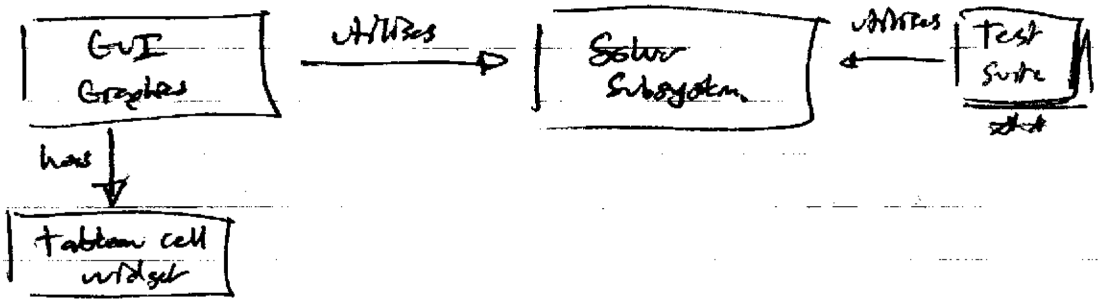
if ui vj found → check optimal.

31 7

TRANSPORTATION TABLEAUX — SPEC.

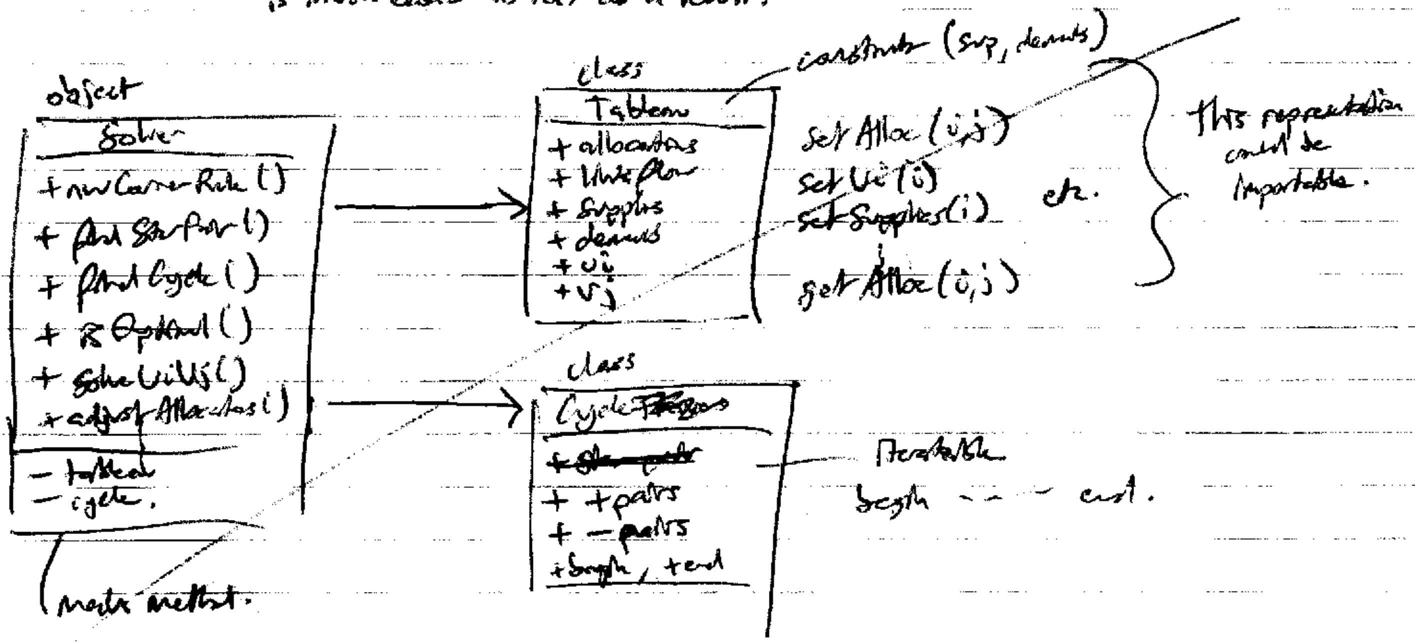
System overview:

- The GUI will be separate from the solver ! this will allow modularity and ease of testing.



Solver subsystem: We will stick with the previous Tableau/CycleTraversal configuration, for the most part. However, we will add a Solver ~~[illegible]~~ class that keeps track of the big picture.

In this way, Tableau becomes more of a data structure class, and is much easier to test as a result.

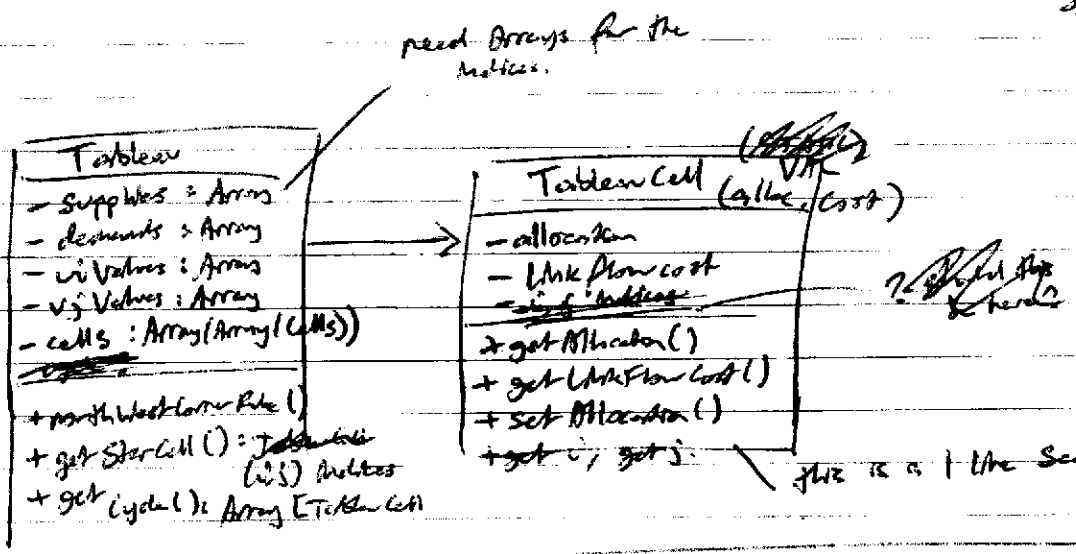


1/3 RESTRUCTURE SUBSYSTEM.

1 8

TRANSPORTATION TABLEAUX — SPEC.

Solver subsystem IDEA — we can restructure the solver to more closely resemble the GUI components, which will ease integration.

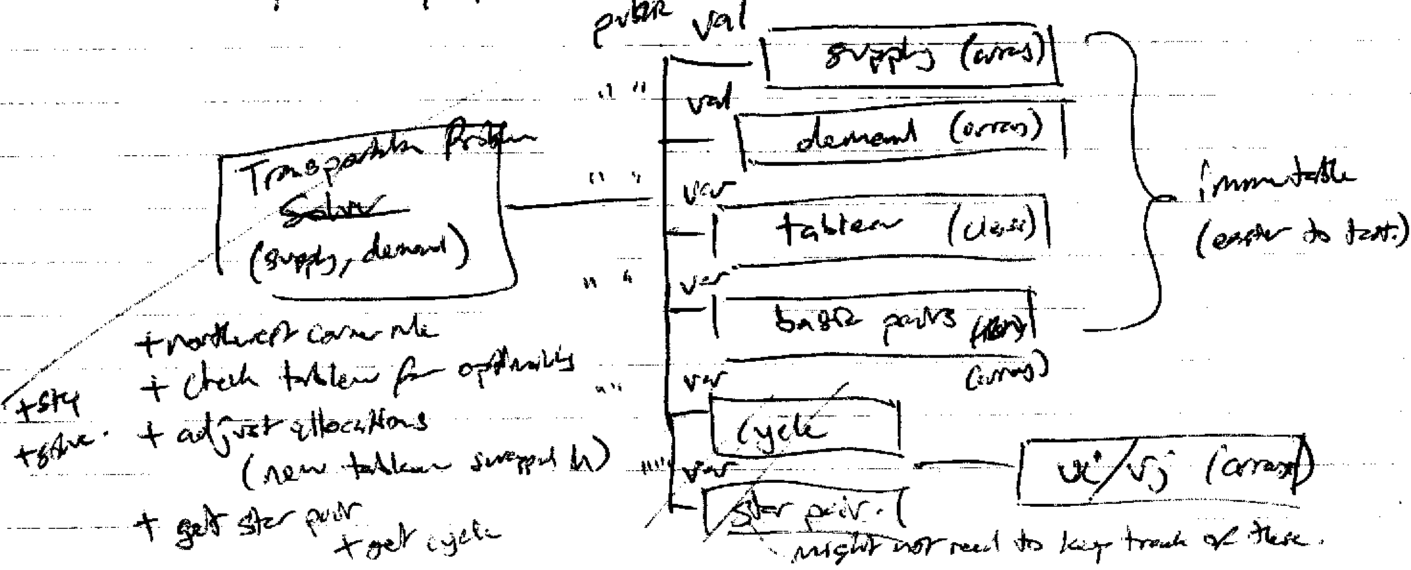


IDEAS:
- ~~tableau has cells~~
- ~~tableau does not need to keep track of +, - pairs in the cycle:~~ In Scala we can efficiently ~~filter~~ them from the traversed order.
(+ → even indices
 – → odd indices)

* The index of each cell is important for drawing the cycle / tableau manoeuvring.
We need to find a way to add this functionality in a simple way.
This ~~for~~ refactoring just seems to be adding complexity though...

2/8 — We need a ~~a~~ representation for:
- tableau (allocation, link flow)
- basic pairs — highlighted in GUI.
- cycle, star pair
- solving the problem
- supply / demand, dual variables ui, vj.

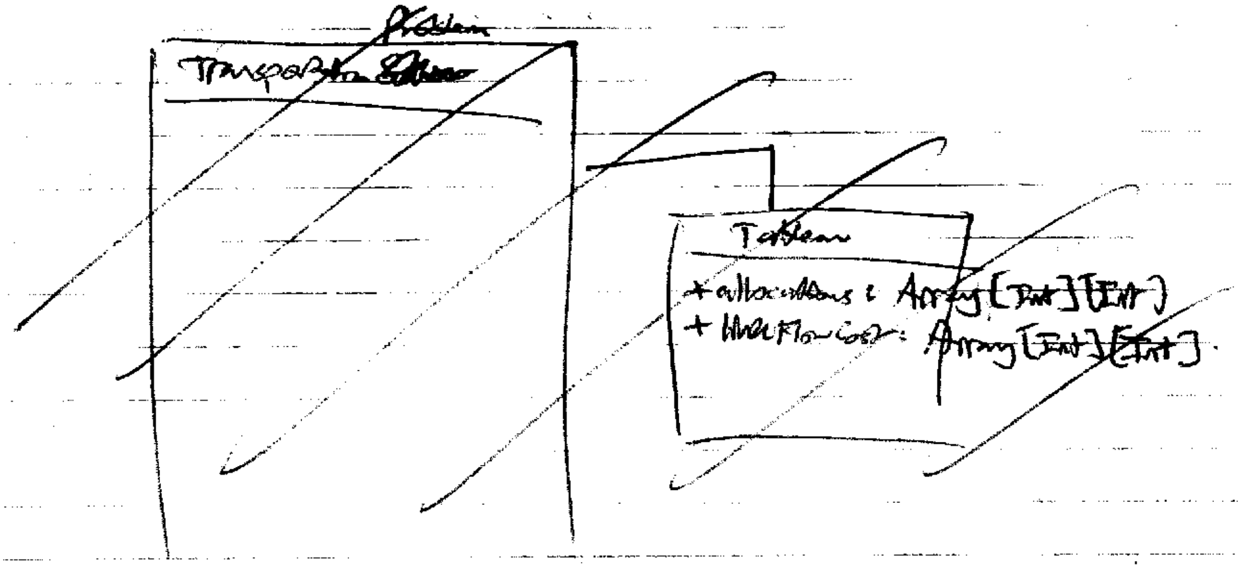
Ideally, tableau should be an immutable ADT that is easy to compare with another tableau for test purposes.



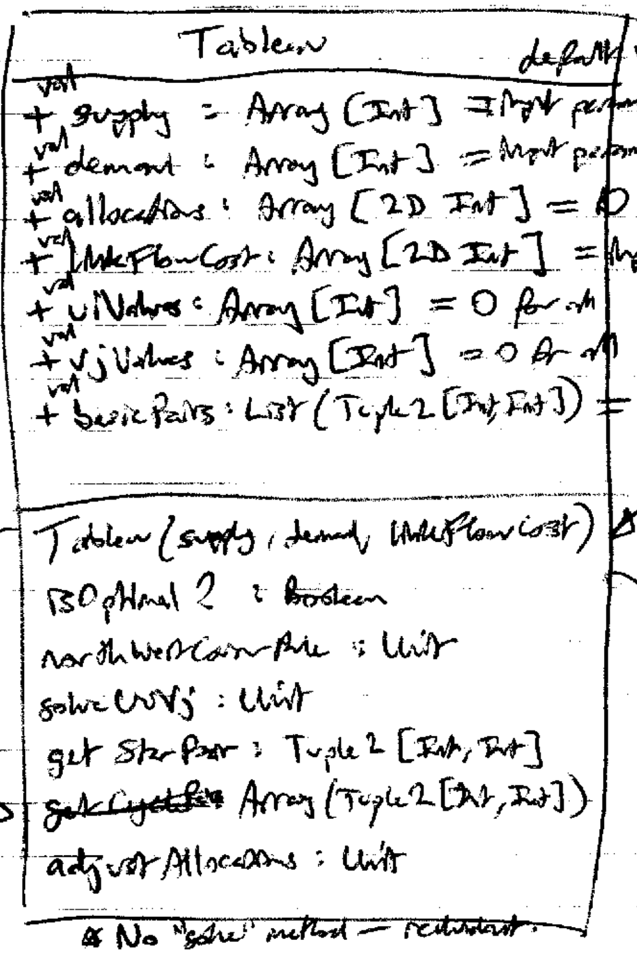
public val
supply (array)
val
demand (array)
var
tableau (class)
var
basic pairs (array)
var
cycle
var
~~star pair~~. (
immutable (easier to test.)
ui/vj (array)
might not need to keep track of these.
Transportation Problem ~~Solver~~ (supply, demand)
+ northwest corner rule
+ check tableau for optimality
+ adjust allocations (new tableau swapped in)
+ get star pair
+ get cycle
+ step
+ solve.

TRANSPORTATIONTABLEAUX — SPEC

Class definitions


Problem
~~TransportationTableau~~
Tableau
+ allocations : Array[Int][Int]
+ unitFlowCost : Array[Int][~~Int~~]

Probably easiest to just put everything in the one Tableau class actually.



```
+ val supply : Array[Int] = input param
+ val demand : Array[Int] = input param
+ val allocations : Array[2D Int] = 0 for all
+ val unitFlowCost : Array[2D Int] = input param
+ val uiValues : Array[Int] = 0 for all
+ val vjValues : Array[Int] = 0 for all
+ val basicPairs : List(Tuple2[Int,Int]) = list of all
```

```
Tableau(supply, demand, unitFlowCost)
isOptimal 2 : Boolean
northWestCornerRule : Unit
solveUiVj : Unit
getStartPair : Tuple2[Int, Int]
~~getCyclePath~~ Array(Tuple2[Int, Int])
adjustAllocations : Unit
```

Represent indexes by Tuple2[Int,Int].

this class will be large and hard to test, so extra discipline will ~~be needed~~

constructor

Code up all tests before constructing each method!

try to use functional paradigm where possible.

getCycleTraversal

* No "solve" method — redundant.

from this, the GUI can access all pertinent values.

* Note that cycle and startpair are now functions (it's not worth keeping these ephemeral values around, and it doesn't make sense to consider them part of the Tableau state.

TRANSPORTATIONTABLEAUX — SPEC.

Still to specify in requirements/design:

☐ · specify complete system overview — for solver subsystem (including algorithm flowcharts)
(Tableau class)

1 · algorithm flowcharts

2 · specify inputs/outputs to all methods/functions.

... (range of values, format, etc.)

3 · specify all tasks that the Tableau should be able to do (including retrieval for GUI viewing purposes.)

☐ · come up with an error detection strategy/plan to handle disk I/O.

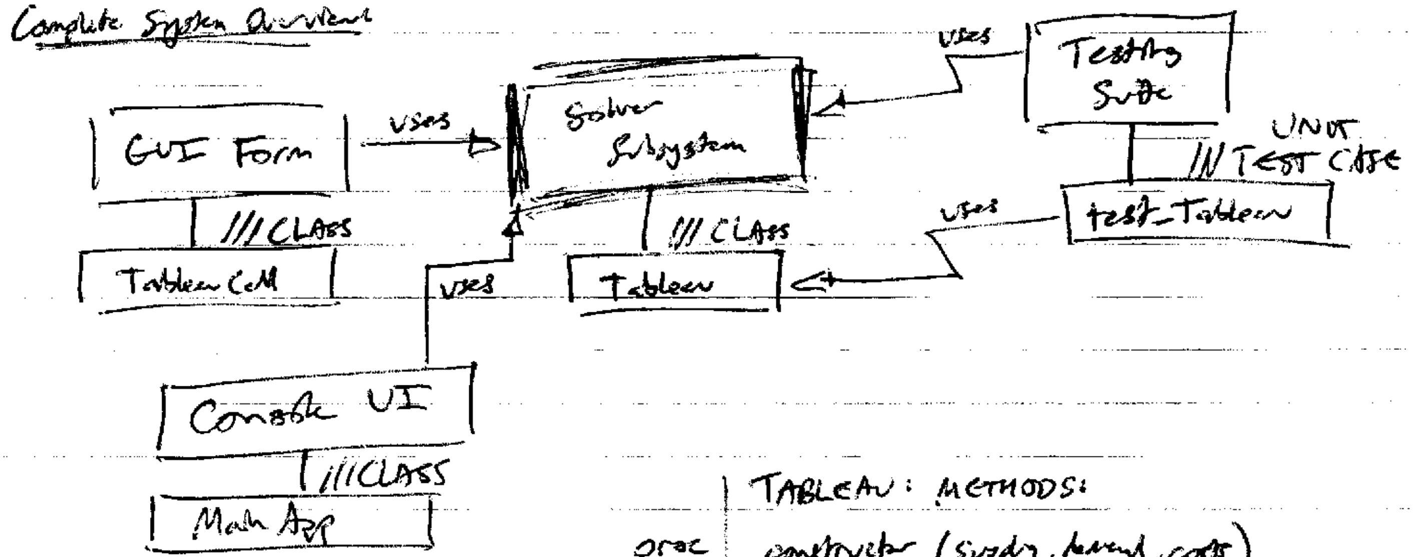
☐ · come up with a test plan for the Tableau (unit testing), as well as a test plan for the integrated system (public test?) * All testing to be done before work on Tableau is started

☐ · design the console test UI —— this should be similar to the existing console UI.

☐ · coding conventions — camelCase as default in Scala.

Need to check OSS Scala projects to see what the community convention is.

Complete System Overview



solve problem as follows:

```
· Tableau (set-up)
· nwCornerRule()
· solveUiVj()
· while (!optHmal) {
-   adjustAllocations (cycleTraversal (startPos))
-   solveUiVj()
}
```

these are functions that do not modify state!

TABLEAU: METHODS:

proc	constructor (supply, demand, costs)
func	isOptimal? (): Boolean
proc	northWestCornerRule (): Unit
proc	solveUiVj (): Unit
* func	~~get~~startPos (): Tuple2[Int, Int]
* func	~~get~~cycleTraversal(startPos from): Array (Tuple2[Int, Int])
proc	adjustAllocations (cycleTraversal): Unit
func	cost(): Int.
val ~~func~~	indices()
~~func~~	~~[illegible]~~

3 8

TRANSPORTATION TABLEAUX — SPEC

Tableau METHODS:

Tableau (supply, demand, linkFlowCost) constructor

1. mySupply ← supply
2. myDemand ← demand
3. myCosts ← linkFlowCost

//assert: size(linkFlow) = size(supply) × size(demand)

1. allocations to 0.
2. ui, vj 0.

4. myAllocations ← New 2D Array [size(supply)][size(demand)]

//assert: size(myAllocations) = size (link Flow)

5. ui~~Values~~ ← new Array[size (supply)]
6. vj~~Values~~ ← new Array [size (demand)]

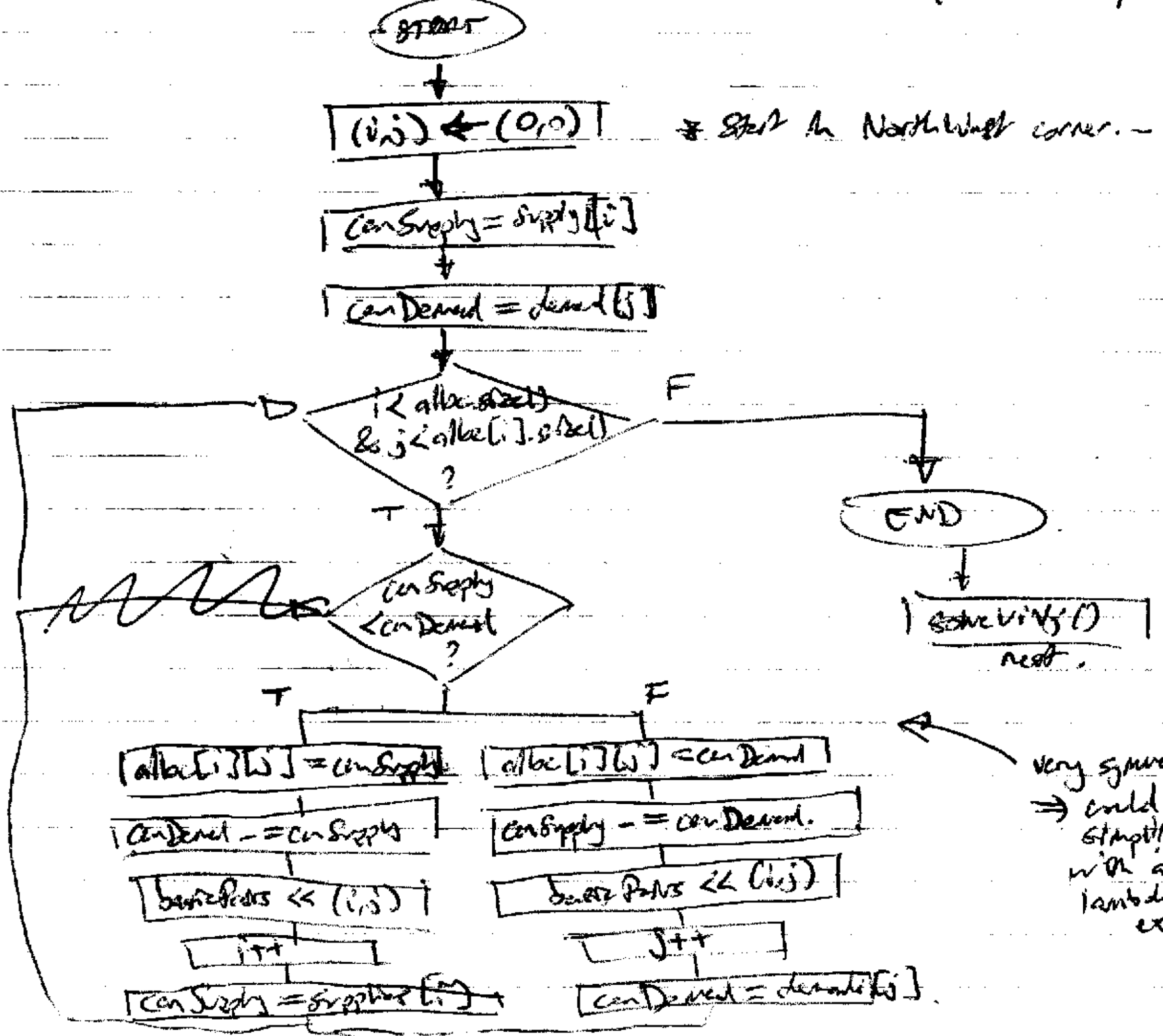
isOptimal ? () : Boolean
  return (ui[i] + vj[j] <= linkFlow[i][j] ∀ i, j.)

add indexPairs method? — returns all (i,j)

(Functional paradigm?)

northWestCornerRule ( ) : Unit

(Procedural paradigm)



* Start in NorthWest corner.

very symmetric
⇒ could we simplify this with a lambda expression?

TRANSPORTATION TABLEAUX — SPEC.

Tableau methods

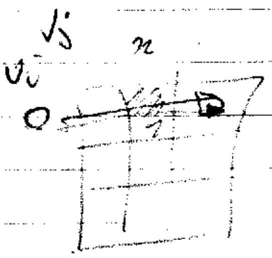
cost() : Int : return sum of (allocations[i][j] × linkFlowCosts[i][j]) ∀ i,j
(Functional paradigm?)

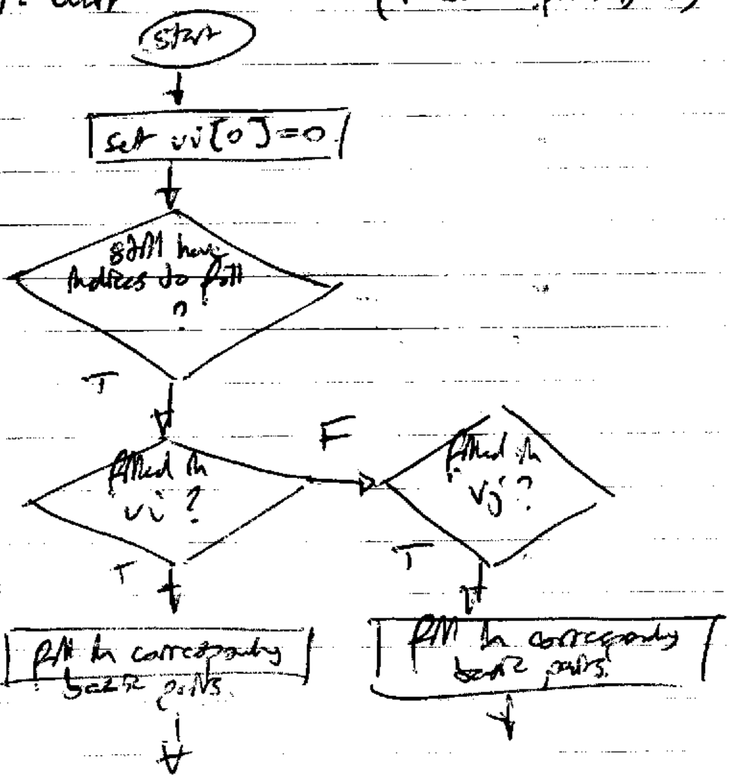
1st attempt at coding these functional methods:

```
def BOptimal(): Boolean =
    indices.forall { (p) => vi ~~viValues~~(p._1) + vj ~~vjValues~~(p._2) <= linkFlowCosts
                                                                   (p._1)(p._2)}

def cost(): Int =
    indices.foldLeft(0) {
        (sum, p) => sum + allocations(p._1)(p._2) * linkFlowCosts(p._1)(p._2)
    }

where ~~def~~ val indices() = for (i <- supply.indices, j <- demand.indices) yield (i,j).
```

solveUiVj() : Unit   (Procedural paradigm?)

```
1. vi[0] = 0
2. while (filled vi or
          filled vj )
3.   if filled vi
       (i,j) = filled vi
4.   if base pairs (x,y)
       x = i and vj[y] not
       set,
       set vj[j]
         = linkFlowCosts[i][j]
           - vi[i]
5. else
     if filled vj
       (i,j) = filled vj
```




start
set vi[0]=0
still have indices to fill?
T
filled in vi?
F
filled in vj?
T
T
fill in corresponding base pairs.
fill in corresponding base pairs.

this algorithm is fairly convoluted, and since the existing one works, we won't spend too much time on refactoring..

(alternatively, there is a lot of symmetric code here too, and we might want to try using some functional recursion/etc. to pretty up the method after all.)

TRANSPORTATION TABLEAUX — SPEC.

adjustAllocations()
CPP Method:

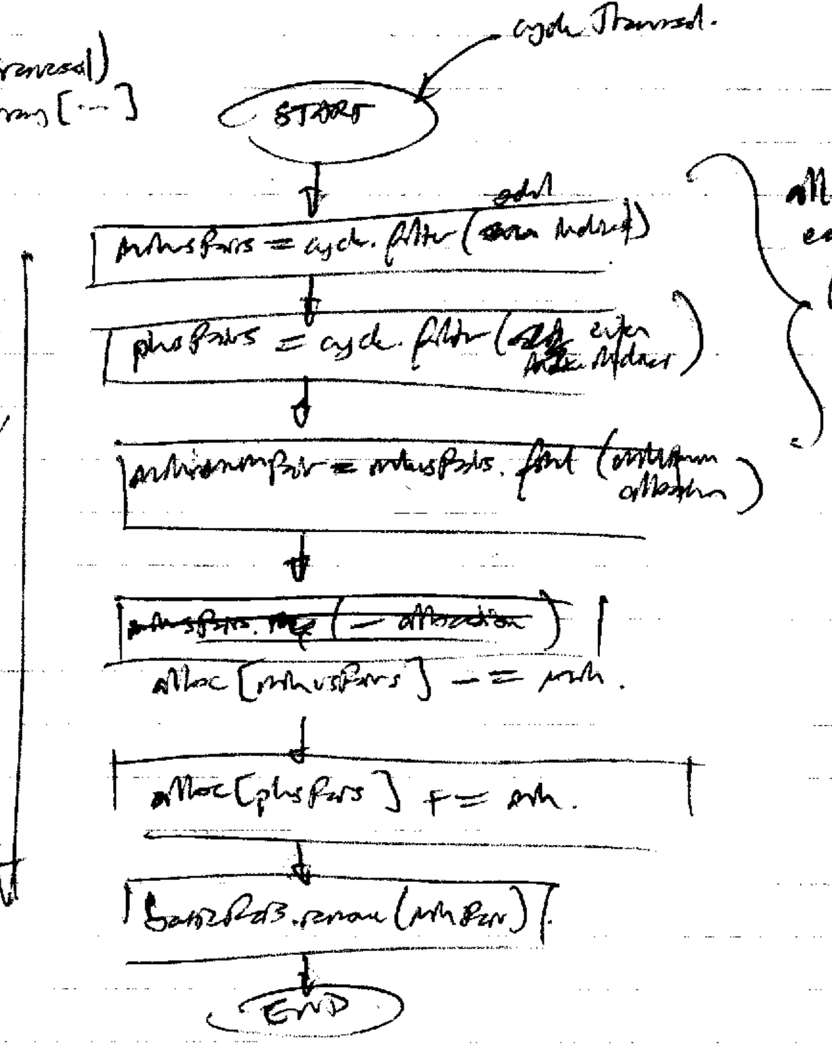
odd ~~even~~ - index 1, 3, 5,

1. get minus pairs from cycle
2. get plus pairs from cycle. — even 0, 2, 4, —
3. get minimum allocation pair in minus pairs

4. In all minus pairs, — minimum
5. In all plus pairs, + minimum
6. remove minimum allocation pair from the basis pairs.

Our new method takes a cycleTraversed as a parameter

adjustAllocations(cycleTraversed)
'Array[...]


cycle Traversed.
START
minusPairs = cycle.filter(odd ~~even~~ index)
plusPairs = cycle.filter(~~odd~~ even index)
minimumPair = minusPairs.find(minimum allocation)
~~minusPairs.map(— allocation)~~
alloc[minusPairs] —= min.
alloc[plusPairs] += min.
basisPairs.remove(minPair)
END

all this is easy with functional paradigm

mostly a straight forward method

Find out how Scala code generally gets documented.

Seems to be mostly similar to Javadoc (probably should stick to my current convention of class usage inclusion though.)

TRANSPORTATIONTABLEAUX — SPEC

starPair()

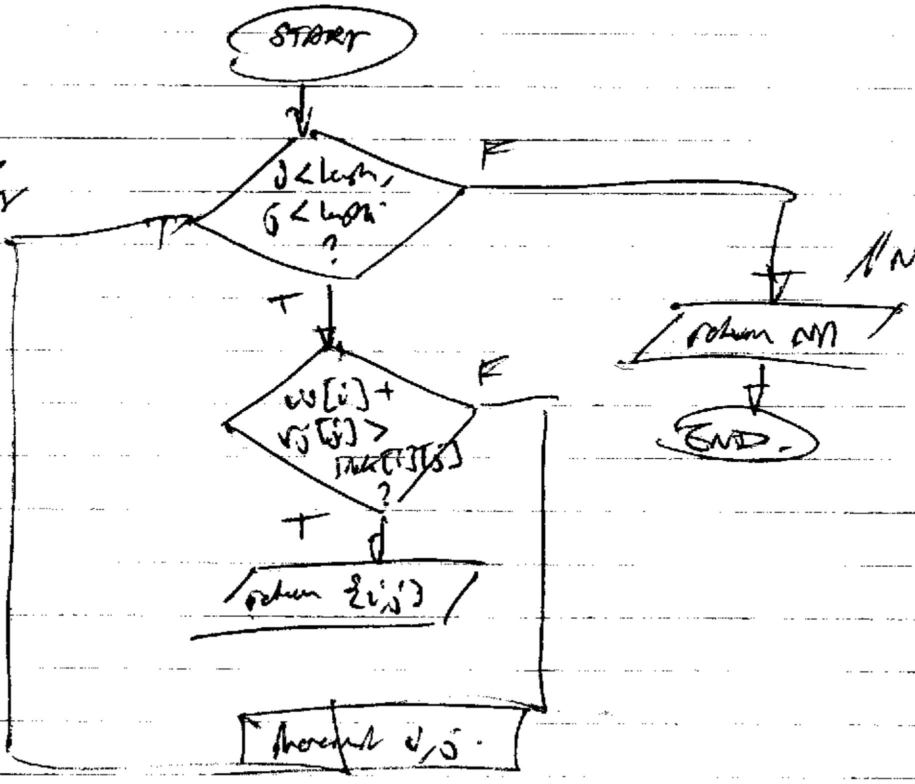
CPP:

1. Iterate over linkFlowCosts
2. If ~~?~~ ui[i]+vj[j] > linkFlow[i][j]
3. star pair is {i,j}.

adds to basic pivot CPP method

Our Scala method will be functional: return star pair, or NIL if not found.

starPair()



~~// Use vectors, comprehensions, not~~

~~Use vectors to store~~

Need to use Arrays for all mutability.

// probably some way to do this functionally.

cycleTraversal()

CPP: // Most complicated method of all.

(Simply copy over the existing logic — already tested.)

findAdjacent — companion object method.

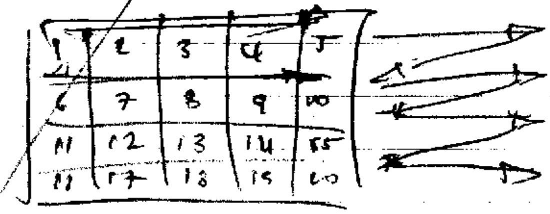
Tableau.cycleTraversal = findCycle + cycleTraversal order.

9 8

TRANSPORTATIONTABLEAUX – SPEC.

* IDEA — perhaps we really should use a Matrix (Grid) ADT to hold the allocations and unit-flow costs?

- Allows us to not worry about the flexibility of choosing Array vs. Vector.
- can define easier accessor/mutator methods. (rows, columns)
- can define an iterator:



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

- Having this iterator allows for the cost, [illegible], isOptimal, etc. methods to be written in a concise and functional manner.

eg.

```
def cost(): Int ~~it~~ =
  indexes.foldLeft(0) {
    (sum, p) => sum + allocations(p._1)(p._2) * unitFlowCosts(p._1)(p._2)
  }
```

```
for indexes = ~~then~~ for (i <- supply.indices; j <- demand) yield (i,j).
```

becomes:

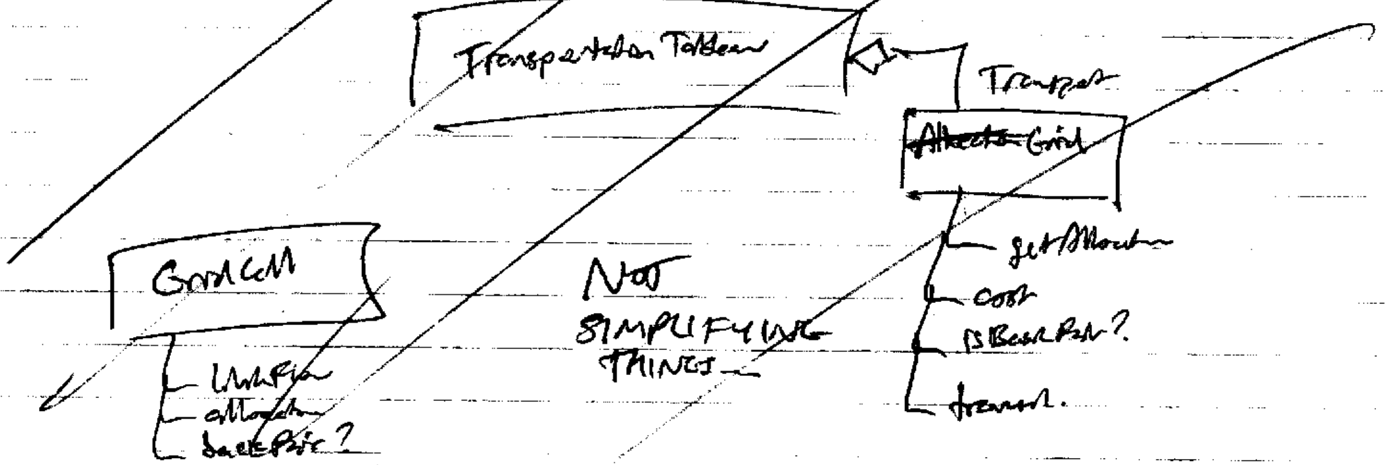
```
def cost(): Int =
```

Doesn't seem to reduce complexity enough to be worth the effort.

Case class TransportationTableau scala

TRANSPORTATIONTABLEAUX — SPEC:

~~For a functional paradigm (and to default to vectors) we should probably use a separate grid object after all.~~

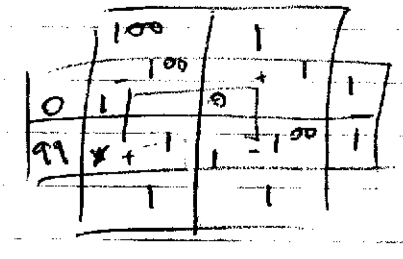


NOT SIMPLIFYING THINGS_

FINISH THE CLASS AS PLANNED.

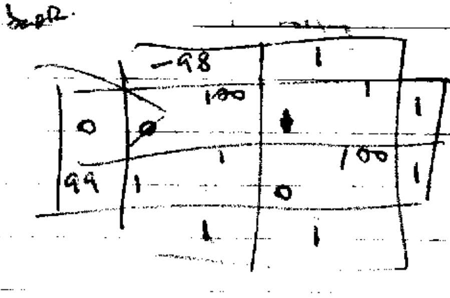
15/3/2014 · Start work on Tableau testing (use Examples from Unit Work).

eg.



try the basic (100/1, 100/1) allocation problem (Note that A B chosen specifically so that the NW corner rule gives the worst case solution. The cycle is always all 4 parts, and the sharpest is always the bottom one in the tableau.)

next basic.



✓

optimal: cost = 2,
alloc: x_{12}, x_{21}.

} put in Scala test.

* Added private var basicPairs(), to be accessed outside of the class with basicSolution.()
Also, adjust allocations cells selectively after it's done.

16/3 —— CLASS FINALISED. ✓ START TESTING ← coded up lecture example test.

· START CONSOLE ~~BASE~~

· START GUI ~~BASE~~

do: { the above allocation problem
· a trivial case (NW gives optimal)
· the 2 homework examples.

sbt — build scala projects
maven/ant — build java projects.

19 8 2014

TRANSPORTATION TABLEAUX — TESTING.

Considerations: Using sbt to build the jar executable — might need to overhaul the directory structure to be compatible with sbt.

```
TransportationTableaux — holds github files, etc. Also build.sbt.
 L src
    L main
    |  L scala
    |
    L test
       L scala
```

.gitignore
should contain target/ (already there)

Tests: Example 1:
We follow the lecture example: * Check indices → works correctly too!

Setup

Initialised to 0.

allocations = 0.

	0	0	0	0	
0	3 0	5 0	7 0	11 0	100
0	1 0	4 0	6 0	3 0	130
0	5 0	3 0	12 0	7 0	170
	150	120	80	50	

* Check that supplies, demands, unit-flow costs got stored correctly.

Initially:
cost = 0
no basic solution
ui = 0
vj = 0
not optimal

After NW Corner Rule

	3	6	10	5	
0	– 3 100	+ 5 *	7	11	100
-2	1 + 50	4 – 80	6	3	130
2	5	8 40	12 80	7 50	170
	150	120	80	50	

cost = 2300
not optimal.

Check stepping path and cycle
(traversed left, right, up, down.

19 8 2004

TRANSPORTATION TABLEAU TESTING

After 1st adjustment:

	3	5	9	4	
0	3 20	- 5 30	+ 7 *	11	100
-2	1 130	4 0	6	3	130
3	5	+ 8 40	- 12 80	7 50	170
	150	120	80	50.	

cost = 2220
not optimal.

After 2nd adjustment

	3	3	7	2	
0	- 3 20	5 ✗	+ 7 80	11	100
-2	1 130	4	6	3	130
5	+ 5 ✗	8 120	- 12 0	7 50	170
	150	120	80	50	

cost = 2060
not optimal.

After third adjustment

	3	6	7	5	
0	- 3 20	+ 5 *	7 80	11	100
-2	1 130	4	6	3	130
2	+ 5 0	- 8 120	12 ✗	7 50	170
	150	120	80	50	

cost = 2060
not optimal.

19 8 2014

TRANSPORTATION TABLEAUX TESTING

After the 4th adjustment:

	2	5	7	4	
0	3 ~~0~~	5 20	7 80	11	100
-1	1 130	4	6	3	130
3	5 20	8 100	12	7 50.	170
	150	120	80	50	

cost = 2040
optimal.

~~END EXAMPLE.~~

Example 2: sup = (1,1), dem = (1,1), tf = (100,1,1,100)
(designed so that NW always gives the worst one.

~~Initial~~ /
NW corner rule.

	100	1	
0	− 100 1	+ 1 0	1
99	+ 1 0 ~~x~~	100 1 −	1
	1	1	

basic variables (0,0),(0,1),(1,1).
cost = 200
not optimal.

after 1st adjustment

	~~99~~ 100	~~4~~ 1	
0	100	1 1	1
~~99~~ -99	1 1	100 ~~0~~	1
	1	1	

basic variable that gets cut from the cycle is always the first found basic var in the cycle!

cost = 2

optimal

END EXAMPLE.

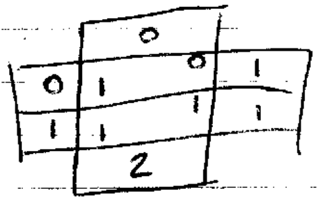
// Implemented both of these tests 20/8/2014.

19 8 2014

TRANSPORTATIONTABLEAUX TESTING.

Example 3 (trivial example) : (sup = (1,1), dem = (2)) 1f = (0,1).

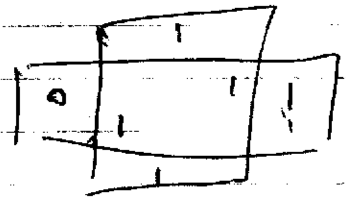
Init = optimal.



cost = 1. TRIVIAL.

END EXAMPLE ——— Coded up on 22/8/2014.

Example 4 (most trivial example) : sup = (1), dem = (1), 1f = (1).



Initial = optimal.

cost = 1.

Basic solution (0,0)

END EXAMPLE. ——— Coded on 25/8/2014.

* Note that the 0-alloc case (i.e. optimal is no allocation) causes an infinite loop on the Scala version of the program. (not in the C++ version though.)
This is an allowable "bug" — what else should the program do in such a situation?

20/8/2014 ——— TODO: further testing (above 2 examples, then add 2 assignment worked examples, then done for now.)

22/8/2014 — finished coding the ConsoleDriver first attempt.
Needed to modify the square width sizes.

25/8/2014

— TODO: testing (2 assignment worked examples, + extra examples as well?)

· fix ConsoleDriver (balance problem...)

Then: start work on Scala.swing GUI.

25 8 2014

TRANSPORTATION TABLEAUX TESTING.

Example 5 (From OOR Assign 4): —— test coded 25/8/2014.

Initial

	0	0	0	
0	3 0	2 0	3 0	9
0	3 0	5 0	1 0	5
0	4 0	6 0	3 0	2
	8	4	4	

cost = 0

not optimal

After Northwest corner rule:

	3	2	-2	
0	− 3 8	+ 2 1	3	9
3	3 * +	5 3 −	1 2	5
5	4	6	3 2	2
	8	4	4	

cost = 49

not optimal.

basic solution
(0,0), (0,1), (1,1), (1,2)
(2,2)

star pair = (1,0)
cycle:
(1,0) → (1,1) → (0,1) → (0,0)

1st adjustment:

	3	2	-2	
0	3 5	2 4	3	9
3	− 3 3	5 ~~0~~	+ 1 2	5
5	4 * +	6	3 2 −	2
	8	4	4	

cost = 40

not optimal.

basic solution (0,0), (0,1),
(1,0), (1,2), (2 2)

star pair = (2,0)
cycle: (2,0) → (2,2) →
(1,2) → (1,0)

2nd adjustment:

	3	2	1	
0	3 5	2 4	3	9
0	3 1	5	1 4	5
1	4 2	6	3 ✗	2
	8	4	4	

cost = 38

optimal.

basic solution =
(0,0), (0,1), (1,0),
(1,2), (2,0)

25 8 2014.

TRANSPORTATION TABLEAUX TESTING

Example 6 (from OOR Assign 4):

Initial

Coded → 28/8/2014.

	0	0	0	0	
0	15 0	20 0	16 0	21 0	250
0	25 0	13 0	5 0	11 0	130
0	15 0	15 0	7 0	17 0	235
	75	240	230	70	

cost = 0

not optimal.

After North Corner Rule

Coded → 27/8/2014

	15	20	12	22	
0	15 75	20 175 −	16 0	21 ~~0~~ +	250
-7	25 0	13 65 +	5 65 −	11 0	130
-5	15 0	15 0	7 165 +	17 70 −	235
	75	240	230	70	

Not optimal

cost = 8140

basic soln: (0,0)(0,1), (1,1), (1,2)
(2,2)(2,3)

steep pivot: (0,3)

cycle: (0,3) → (0,1) → (1,1),
→ (1,2) → (2,2) → (2,3)

After 1st adjustment

Coded → 30/8/2014

	15	20	11	21	
0	15 75	+ 20 110	16	21 65 −	250
-7	25	13 130 −	5	11 ~~0~~ +	130
-4	15	15	7 230	17 5	235
	75	240	230	70	

Not optimal.

cost = 8075

soln: (0,0), (0,1), (0,3), (1,1),
(2,2), (2,3).

steep path (1, 3)

cycle: (1,3) → (1,1) → (0,1) → (0,3)

After 2nd adjustment

Coded 30/3/2014 ↳

	15	20	8	18	
0	15 75	20 175	16	21	250
-7	25	− 13 65	5	11 65 +	130
-7	15	15 ~~0~~ +	7 230	17 5 −	235
	75	240	230	70	

Not optimal

cost = 7880.

soln: (0,0), (0,1), (1,1), (1,3),
(2,2), (2,3).

steep path (2,1)

cycle (2,1) → (2,3) → (1,3) → (1,1)

After 3rd adjustment

Coded → 30/3/2014

	15	20	12	18	
0	15 75	20 175	16	21	250
-7	25	13 60	5	11 70	130
-5	15	15 5	7 230	17	235
	75	240	230	70	

cost = 7860

soln (0,0), (0,1), (1,1), (1,3)
(2,1), (2,2).

optimal.

2 9 2014

TRANSPORTATION TABLEAUX GUI.

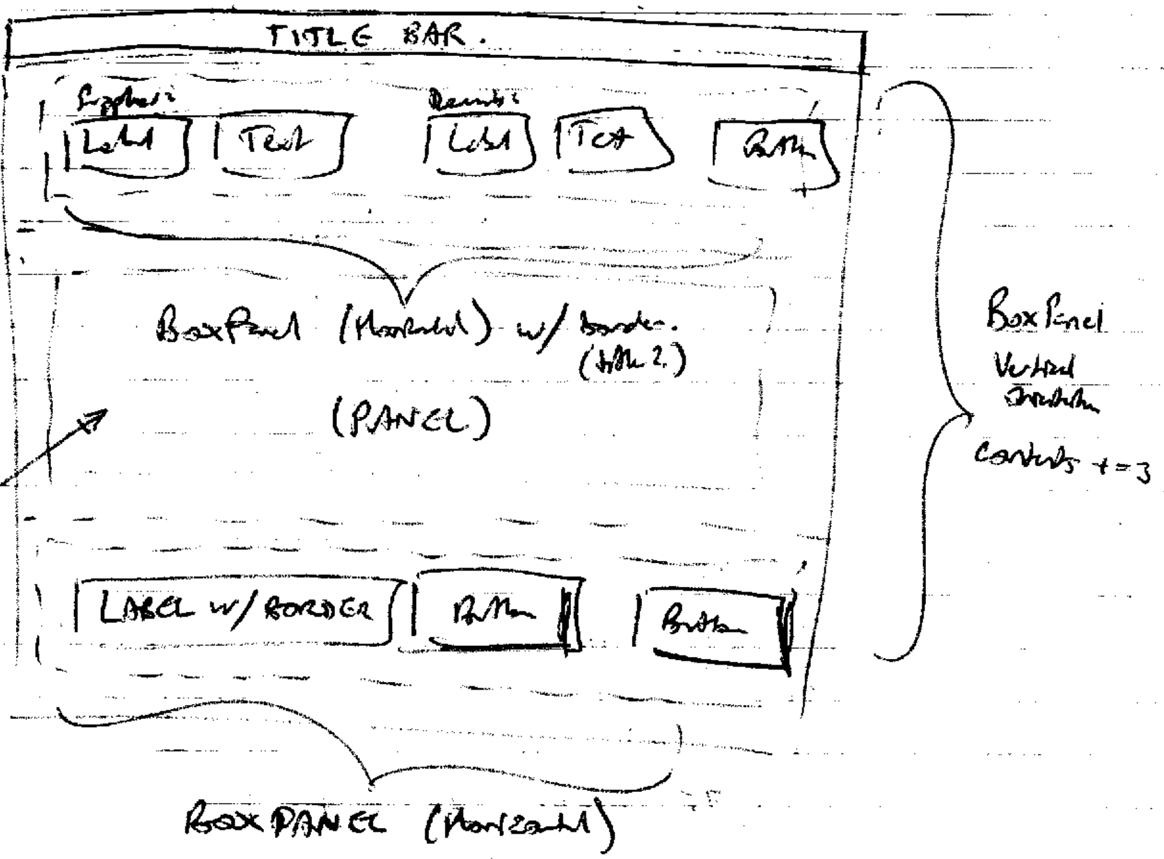
Still need to:

- modify console to balance transportation problem (with prompt.)

* In GUI — prompt when the problem is not balanced. (it might have been in error)
  ie. prompt for whether to add fictitious supplies/demands.

GUI

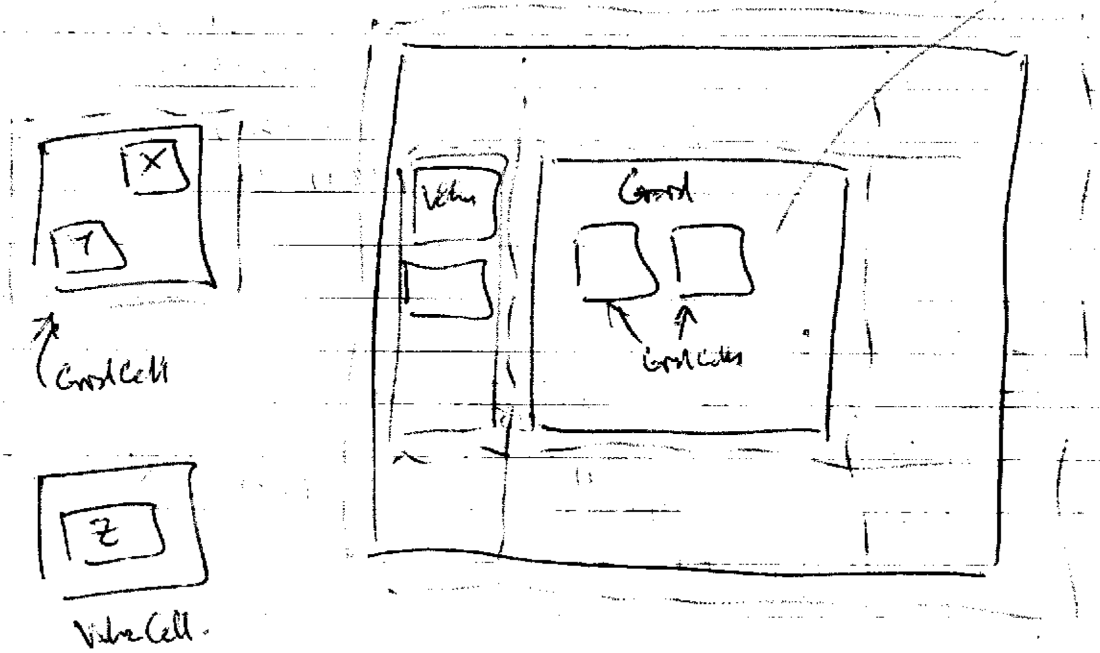
< SimpleGUIApplication.
title="TransportationTableaux"



This will need to be its own widget.

Create separate widget for the Grid.

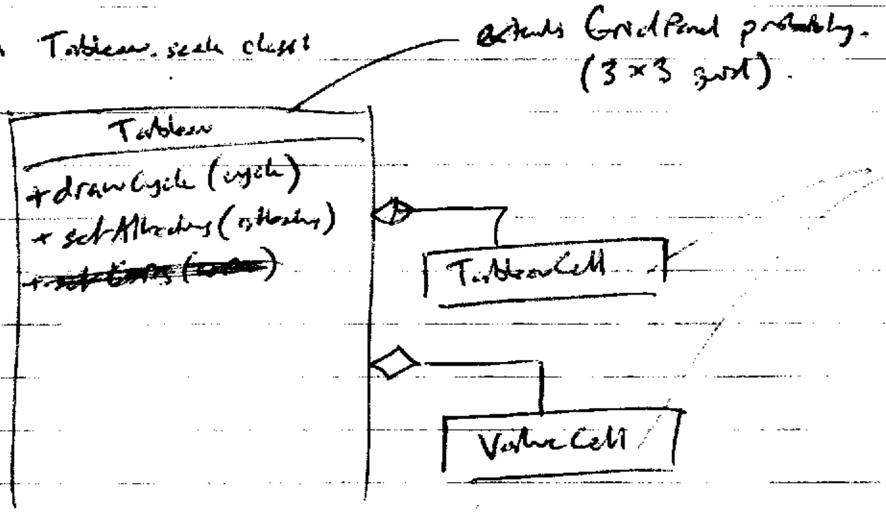
want a Grid object that we can call
. drawCycle (cycle)
on.



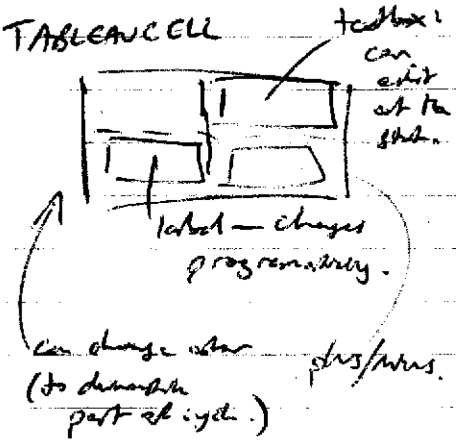
2 9 2014

TRANSPORTATIONTABLEAUX GUI

What a Tableau.scala class:


extends GridPanel probably.
(3×3 grid).
Tableau
+drawCycle (cycle)
+ setAllocations (allocations)
~~+ setCosts (costs)~~
TableauCell
ValueCell
these are classes in their own files.

TABLEAUCELL

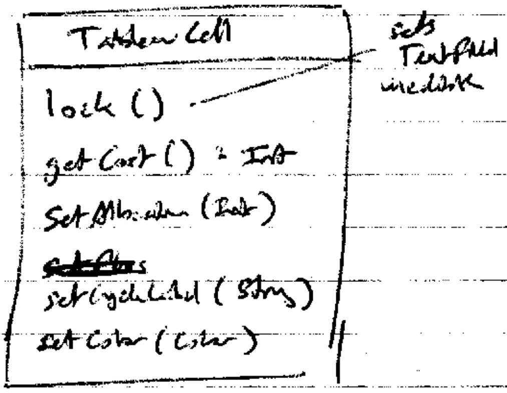

textbox: can edit at the start.
label — changes programmatically.
can change color (to demarcate part of cycle.)
plus/minus.

Code:

```
class TableauCell extends GridPanel(2,2) {
    private val allocation = new Label("0")
    private val unitFlowCost = new TextField()
    private val cycleLabel = new Label("")

    maximumSize = new Dimension(64, 32)
    ~~background~~
    border = LineBorder(Color.BLACK)
    contents += new Label("")
    contents += unitFlowCost
    contents += allocation
    contents += cycleLabel

    def setColor(color: java.awt.Color) {
       background = color
    }

       lockIn()
    def ~~lock()~~ { unitFlowCost.editable = false }


    def getCost() = unitFlowCost.toInt.
                        value
    def setAllocation(~~allocation~~ Int) {
        allocation = value
    }

    def setCycleLabel(label: String) {
        cycleLabel = label
    }
```

change these as necessary!

blank spot in top-left corner.

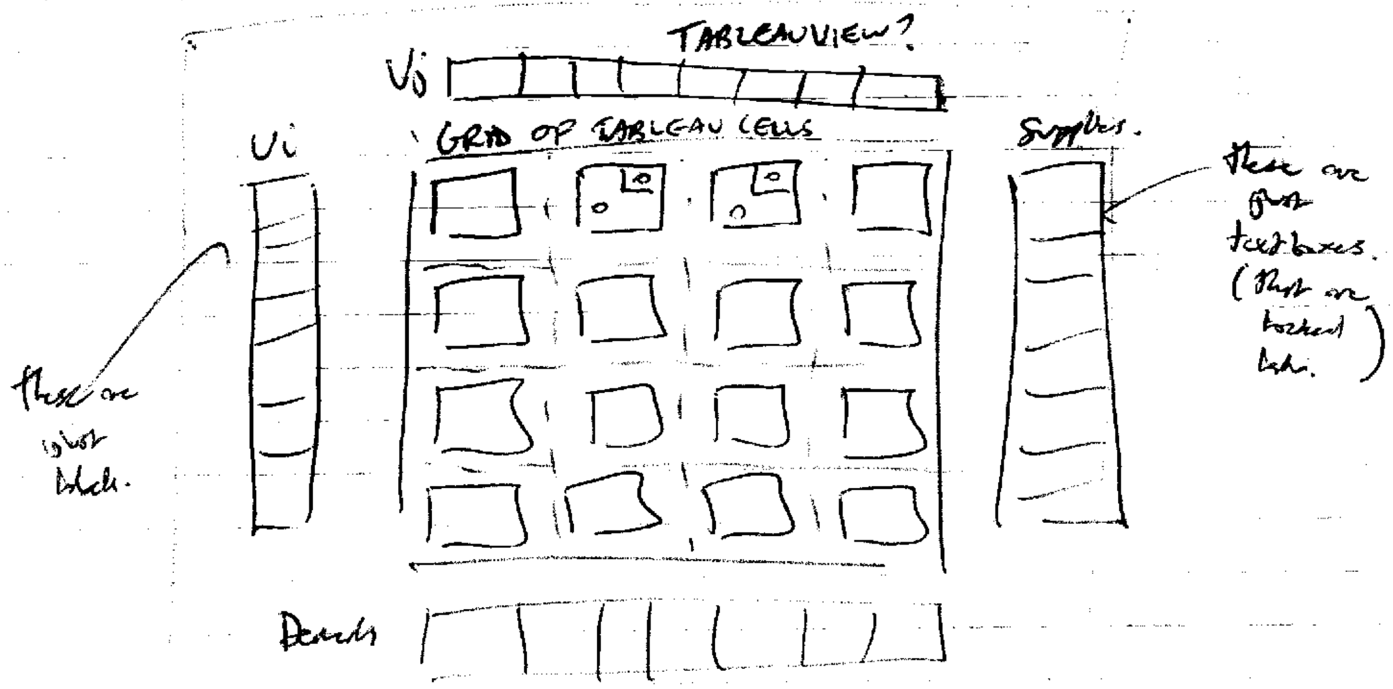


- finish GUI } 1st.
- finish Cascade }
- package into jar. — 2nd (last.)

3 9 2014

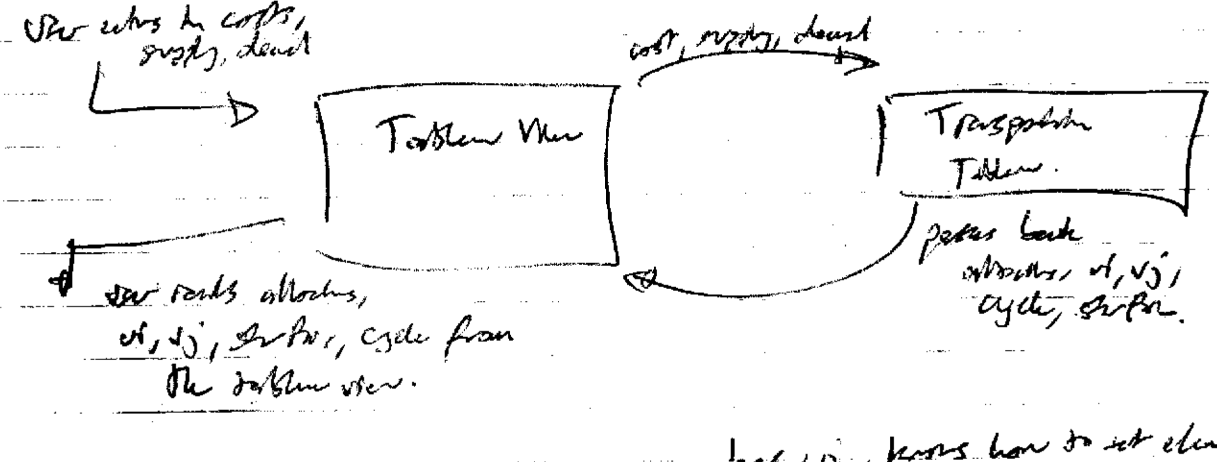
TRANSPORTATION TABLEAUX GUI

* Ideally, we want a "Tableau" object that we can update easily by just passing it the results of the allocations.



TableauView — setUi(~~ArrayList()~~ Array), setVj(~~ArrayList()~~ Array),
setAllocations(~~ArrayList(ArrayList())~~ Array Array), setCycle(),
setStonePair(),

getSupplies(), getDemands(), getCosts()


User enters in costs, supply, demand
cost, supply, demand
TableauView
Transportation Tableau.
passes back allocations, ui, vj, cycle, stonePair.

~~set~~ results allocations, ui, vj, stonePair, cycle from the tableau view.

* ~~A~~ TableauView will be a composite object
 ↳ SuppliesView } textbox arrays.
 ↳ DemandsView
 ↳ UiView } label arrays
 ↳ VjView
 ↳ UnitFlowGrid — this is a big class.

has Ui, knows how to set elements
has Vj, knows how to set elements.
has allocations, knows how to set elements
has Grid().
knows how to set a stone pair
has Grid().
knows how to draw a cycle.

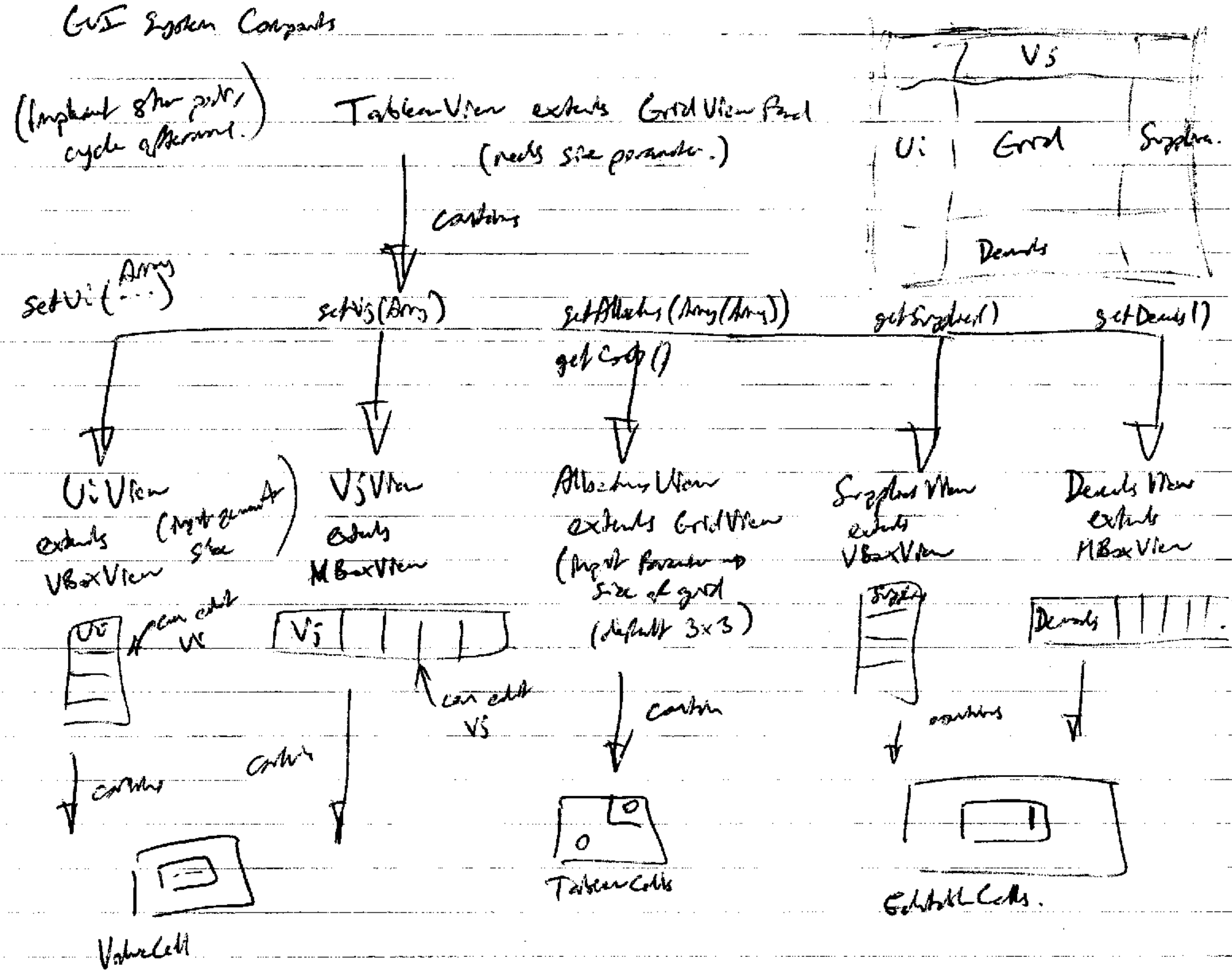
* Fixed up some bugs in the TransportationTableau documentation.

5 9 2014

TRANSPORTATIONTABLEAUX GUI

GUI System Components

(Implement other parts, cycle afterward.)



* Simple Swing Application instead of SimpleGUIApplication

* To run a Scala Swing App:

```
scalac -classpath /usr/share/java/scala-swing.jar ______ .scala

then scala -classpath /usr/share/java/scala-swing.jar ______
```

↑ not the scala file!

Names:

~~LinkFlowCell~~

~~Grid~~ ~~GridCellView~~ ✓

~~ValueCellView~~ ✓

~~CellView~~

~~Editable CellView~~

~~Edit~~

GridCell ✓

ValueCell

EditableCell ✓

Don't worry about sizes of grid cells!

UiView

VjView

~~Grid~~ LinkFlowView

SuppliesView

DemandsView.

} TableauView.

5 9 2014

TRANSPORTATIONTABLEAUX GUI

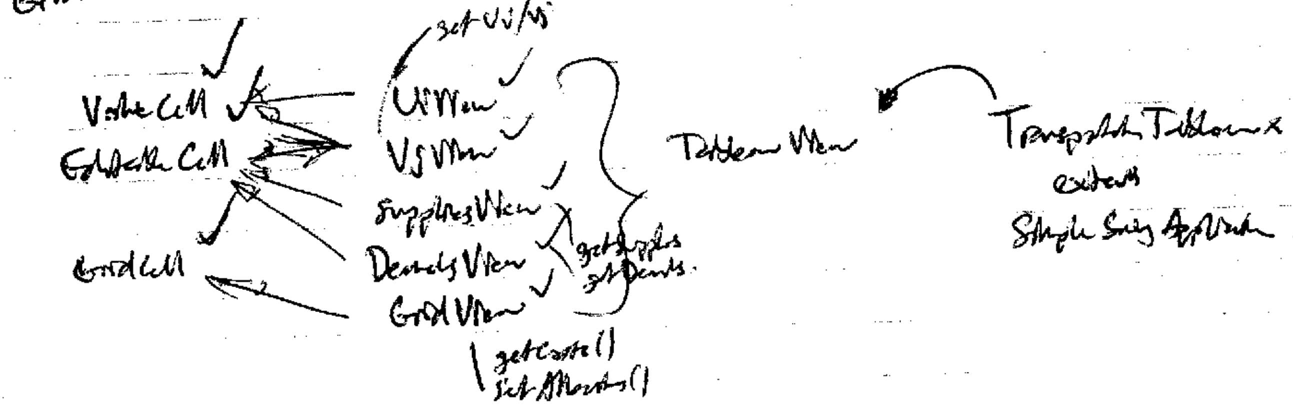
#Finished prototypes for the GridCell, ValueCell, EditableCell classes.

Next Ui, Vj View → take in length (from problem size)
filled with value cells
~~[illegible] set individual ui~~
sets all ui from a passed in array. (from tableau.)

Sup, Dem View → " same, except editable cells, and returns an array of the integers for their values.

GridView started.

Classes

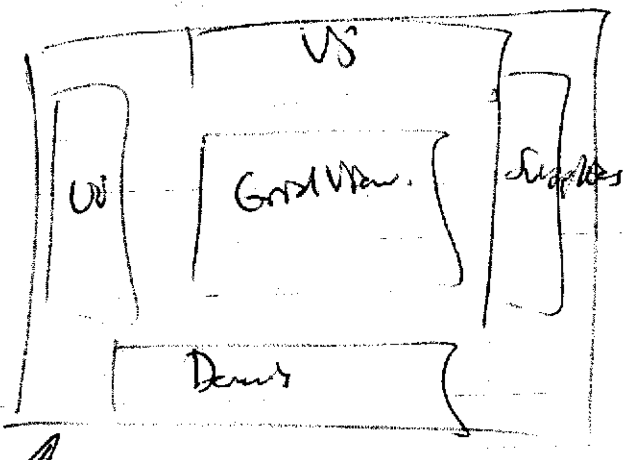


Then : TableauView — passes along arrays to each of its components.

extends BorderPanel.

~~takes in~~ supplyCount, demandCount parameters.

Can set Ui, Vj, allocations.
get Sup, dem, costs.



5/9/2014

The margins for the BorderPanel don't quite work as intended.
Will need to look into this.

6/9/2014 Added licenses :

6 9 2014

TRANSPORTATIONTABLEAUX

* At some point in the future: go through and rewrite according to Scala User Style Idioms:

- replace { } with ~~Unit~~ : Unit = { } in method declarations.
- curly-braces on if, not on for (we use on if clause!) ✓
- on for, choose to curly braces for yield clause generators though.

Done. Also added SBT Build configurations for Swing.

I seem to have gotten the layout to work, but the grid expansion could be tricky.

* (Also, need to do User Guide in LaTeX for documentation.)

8/9/2014 — Finished alpha build.

Next steps: Need to refactor GUI classes / clean up.

- try to get tabbing working properly (tabs should highlight entry!)
- work on getting the resize working okay
- after 'Step' or 'Solve' clicked, shouldn't be able to change numbers.

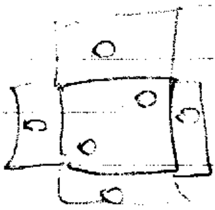
Then:
- get StatusBar showing up
- get ~~cycle draw~~ cycle drawing working.

13/9/2014

- cleaning up the ConsoleDriver.scala class ✓

→ add Documentation: version 1.0

Still to do: Document everything.



* Changed TransportationTableau.scala so it handles the pathological 0-0 case (which we consider optimal from the start.
Still doesn't handle 0-supplies, 0-demands, but I think this is fine — this case should never happen.

Test of pathological case: ✓

- should be optimal
- should provide basic solution () (empty)

13 9 2014

TRANSPORTATION TABLEAUX

More tests to be added:

- Tutorial 4, Question 1 example 8
- Tutorial 4, Question 3 example 9
- ~~2012 Exam, Question 1 (b) example 10.~~

Tutorial 4, Question 1

supplies: 175 75 200
demands: 100 50 100 200
costs: 14 10 12 8 7 7 10 9 13 10 4 9

Initial + NW corner rule.

Start corner (2,3)

cycle: (0,3)→(0,2), →(1,2)→(1,3).

	14	10	12	11	
~~0~~	14 100	10 50	12 − 25	8 * +	175
−2	7	7	10 75 +	9 0 −	75
−2	13	10	4	9 200	200
	100	50	100	200	

Note: Program always goes to the right in a cycle!

basic solution: (0,0), (0,1), (0,2), (1,2), (1,3), (2,3).

cost: 4750.

Simply changes the basic pairs.

After cost adjustment

Start pair (1,0)

cycle: (1,0)→(1,2) →(0,2)→(0,0).

	14	10	12	8	
0	14 100 −	10 50	12 + 25	8 0	175
−2	7 * +	7	10 75 −	9 ~~0~~	75
1	13	10	4	9 200	200
	100	50	100	200	

basic solution (0,0), (0,1), (0,2), (0,3), (1,2), (2,3).

cost 4750

After 2nd adjustment.

Start corner (2,0)

(2,0) (2,3) (0,3) (0,0).

	14	10	12	8	
0	14 25 −	10 50	12 100	8 0 +	175
−7	7 75	7	10 ~~0~~	9	75
1	13 * +	10	4	9 200 −	200
	~~100~~	50	100	200	

basic solution (0,0) (0,1) (0,2) (0,3) (1,0) (2,3)

cost 4375.

*Coded up on 15/9/2014.

continued

13 9 2014

Still to code: TRANSPORTATION TABLEAUX

Tutorial 4, Question 1 test.

After 3rd adjustment.

Star pair (2,1)

cycle
$(2,1)\to(2,3)\to(0,3)\to(0,1)$.

	12	10	12	8	
0	14 ⊗	10 − 50	12 100	8 25 +	175
-5	7 75	7	10	9	75
1	13 25	10 * +	4	9 175 −	200
	100	50	100	200	

basic solution
(0,1),(0,2),(0,3)
(1,0),(2,0),(2,3)

cost 4325

After 4th adjustment

star pair (2,2)

cycle:
$(2,2)\to(2,3)\to(0,3)\to(0,2)$.

	12	9	12	8	
0	14	10 ⊗	12 − 100	8 + 75	175
-5	7 75	7	10	9	75
1	13 25	10 50	4 * +	9 125 −	200
	100	50	100	200	

basic solution
(0,2),(0,3),(1,0)
(2,0)(2,1),(2,3)

cost: 4275

After 5th adjustment.

	12	9	3	8	
0	14	10	12 ⊗	8 175	175
-5	7 75	7	10	9	75
1	13 25	10 50	4 100	9 25	200
	100	50	100	200	

basic solution
(0,3),(1,0)(2,0),
(2,1)(2,2),(2,3)

cost: 3375.

✓ optimal.

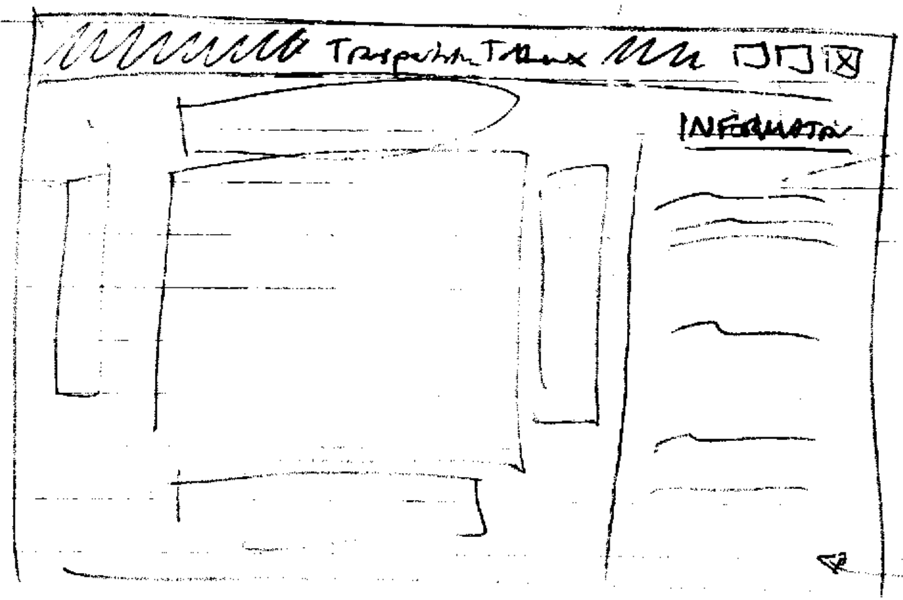
coded up on 16/9/2014.

(More tests: Example 9 — Tutorial 4, Question 3
Example 10 — 2012 Exam, Question 6 b).

13 9 2014

TRANSPORTATION TABLEAUX

* New idea for the GUI : have separate pane on the right to show instructions / other information?



balanced ?
☑ ☒

cost —

basic solution
(in terms of x_{ij})

· optimal?

⋮

would we still need a status, or could we put it in here?

* 13/9 — Cleaned up + documented EditableCell, GridCell ; ~~the~~ ValueCell classes.

Need a way to store the information and status strings...

14/9 — cleaned up the UiView and VjView classes, and the GridView.
Refactored the cell classes.

15/16/9 — testing (Examples & tests.)

For GUI, main things

1. need to represent the star pair (modify grid cell to display (*) in corner.) (method in TableauView.
2. need to represent the cycle ~~(separate CycleDrawer class?)~~ method in TableauView. (draw square cycle, + paint cells?
3. need to show basic solution. (in math style, i.e. $x_{11} = 20, -$ etc) and total cost somewhere on the screen.

Current
1. NW corner
2. adjust
3 adjust again
⋮
n
n+1 optimal.

needs to be →

Final
1. NW corner
{ 2. star pair
3. cycle
4. adjust }
repeat
⋮
n+1 optimal.

21 9 2014

TRANSPORTATION TABLEAUX

2/9 — clean up demandsview, supplyview ✓ ✓

— clean up tableauview. ✓

— example 9 tests. ✓

— cells — highlight when focused?
*added editablecell, gridcell.

This took a surprisingly long time...

(later, error-checking code can go in the same reaction class.)

* LOTS OF DUPLICATE CODE IN:
(uiView / vjView) (demandView / supplyView)

only differences:
- border title
- orientation of box plot.
- name of getter method (supplies/demands)
↳ common getValues() would suffice.)

~~TODO:~~
~~Example 10 tests~~
- rename ~~TestGUI~~ → GuiDriver.

~~REFACTORING:~~
~~Convert Ui/VjView into~~
(DisplayCellView parameterised by:
- title
- orientation?)

this could be an idea (after the program works fully!)

Example 9 tests:
Tutorial 4, Q3

Supplies [15, 17, 18]
demands [18, 15, 10, 7]
tableauCosts [1, 5, 3, 6 3, 2, 6, 2 6, 3, 7, 5]

Initial tableau

coded on 22/9

	0	0	0	0	
0	1 0	5 0	3 0	6 0	15
0	3 0	2 0	6 0	2 0	17
0	6 0	3 0	7 0	5 0	18
	18	15	10	7	

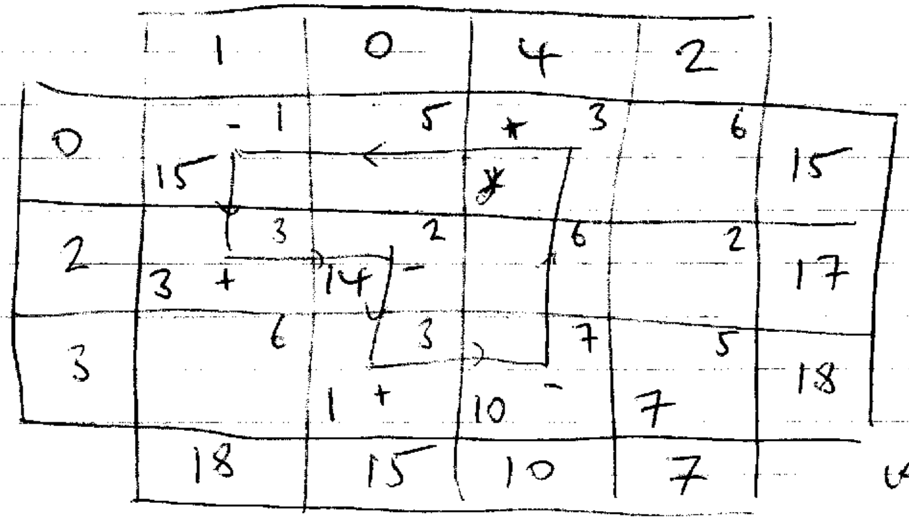
cost 0

Basic cells:
()

After N-W corner rule.

Star Pair: (0,2)

Cycle:
(0,2)→(0,0)→(1,0)
→(1,1)→(2,1)
→(2,2)



	1	0	4	2	
0	- 1 15	5	* 3 *	6	15
2	3 3 +	2 14 -	6	2	17
3	6	3 1 +	7 10 -	5 7	18
	18	15	10	7	

Basic cells:
(0,0), (1,0),
(1,1), (2,1),
(2,2), (2,3).

cost: 160

coded 22/9.

21 9 2014

TRANSPORTATIONTABLEAUX

Example 9 steps:

After 1st adjustment

startPos: (1,3)

cycle: (1,3)→(1,1)→(2,1)→(2,3)

	1	0	3	2	
0	1 5	5	3 10	6	15
2	3 13	−2 4	< 6	+ 2 ⊗	17
3	6	3 11 +	7 ⊗	5 7 −	18
	18	15	10	7	

cost 150

basic solution (0,0), (0,2), (1,0), (1,1), (2,1), (2,3).

coded 23/9.

After 2nd adjustment

startPos (2,2)

cycle: (2,2)→(2,3)→(1,3)→(1,0)→(0,0)→(0,2)

	1	−2	3	0	
0	+ 1 5	5	− 3 10	6	15
2	3 13 −	2 ⊗	6	2 4 +	17
5	6	3 15	7 ✱ +	5 3 −	18
	18	15	10	7	

this loop should be satisfying to code. (If this works in the GUI, use this as the project screenshot!)

cost 142.

basic soln: (0,0), (0,2), (1,0), (1,3), (2,1), (2,3)

coded 19/11.

After 3rd adjustment.

	1	−1	3	0	
0	1 8	5	3 7	6	15
2	3 10	2	6	2 7	17
4	6	3 15	7 3	5 ⊗	18
	18	15	10	7	

cost 139

basic solution (0,0), (0,2), (1,0), (1,3), (2,1), (2,2)

✓ optimal.

coded 19/11.

22 9 2014

TRANSPORTATION TABLEAUX

1/ ✓

+ clearStarPair.

IMPLEMENT STARPAIR:

· adjust GridCell with setStar~~Pair~~ (~~boolean~~) method
()

adjust GridView with setStar~~Pair~~ ((i,j)) methods.
and clearStarPair ()

· adjust TableauView with setStarPair (i,j) methods.
clearStarPair()

(Pertinent classes: GridCell, GridView, TableauView).

should be a straightforward change

true, changes "" to "*" in top-left corner.
false, ↑ same, except "*" → "".

TableauView. setStarPair (i,j)
TableauView. clearStarPair ()
↳ GridView. setStarPair (i,j)
GridView. clearStarPair ()
↳ GridCell. setStarPair ()
GridCell. clearStarPair ()

Also modify testGUI to check this.

3/ ✓

IMPLEMENT CYCLE: — will need to check drawing facilities

Pertinent classes: GridView, TableauView.

* Can use Graphics2D drawing facilities.

· will need: a method that takes in a pair, returns the (x,y) coordinates of the middle of that cell.
in grid view.
a method to draw lines between two cells.

· will need to override paintComponent.

Use a Glass Pane (separate class...) CyclePane class?

✓

2/ (IMPLEMENT +/- ALLOCATIONS ALONG WITH CYCLE?)

Pertinent classes: GridCell ✓, GridView ✓, TableauView ✓.
↑
store in left ~~bottom~~ ~~right~~ corner

set this up, along with colour-changing of cycle pairs. Then we can easily check whether the drawing facility works.

GridCell. setCycle ("+", "-") + change background colour.
GridCell. clearCycle ()
GridView. setCycle (cycle)
GridView. clearCycle ()

CyclePane
CyclePane (cycle: List,

TODO: fix up javadoc on everything.
(what needs to be javadoc and what doesn't (private?)).

25 9 2014

TRANSPORTATION TABLEAUX!

CYCLE IMPLEMENTATION COMPLETE!

= later (after release 0.3).

left to do:

REFACTOR

# · UIView/VGView into one class. SupplyView/DemandView into one class.

✓ · remove cycle Debug info.

✓ · change imports to only what is actually needed.

✓ · rewrite TestGUI → GuiDriver class.

# · refactor CycleView class to be simpler.

TESTING

✓# · example 9 tests

~~# example~~

DOCUMENTATION ✓

✓ · Document

✓ · javadoc all necessary classes (Views, etc.)

✓ · update all version numbers on classes.

(print some documentation for portfolio?)

GUI MODIFICATION (DONE IN GuiDriver, not TestGUI!)

✓ · add section at bottom for tableau cost, basic solution. ✓

# · (optional) — add dialog for unbalanced problem?).

# · figure out if we can solve that "2-click to properly adjust size" bug.

# · (optional) — sometimes the "Found the goal" message doesn't display.
(this might fix itself once we refactor properly however.)

# · fix up the sizing issues on the composite views

RELEASE + USER GUIDE

✓ · complete short tutorial-style guide in latex.

✓ · release version 0.3 (not quite fully done — need beta testers!)

✓ · compile finished assembly (don't worry about size)

# make size smaller?

✓ · DONE!

✓ · updated readme.

26 9 2014

TRANSPORTATION TABLEAUX

Implementing the cost — basic solution functionality.
- the cost was straightforward
- for string for basic solution:
  - need subscript indices: $x_{1,1}$, etc. ✓
  - need to sort basic solution ✓

Later: change this to a list of ~~labels~~ instead? (So it renders better on the FlowPanel..)

NEED TO REFACTOR INTO GuiDriver.scala. DONE.
(might need to work on sizing..)

(To get resizing working properly.)

need to make sure UiView, etc and gridView stick to their proper sizes!

27/9.

Documenting all classes.
(fix up imports?) yes.

GridCell.scala ✓
ValueCell.scala ✓
EditableCell.scala ✓

DemandView.scala ✓
SupplyView.scala ✓
UiView.scala ✓
VSView.scala ✓

GridView.scala ✓
TableauView.scala ✓
Still need to refactor later! → CycleView.scala ✓

ConsoleDriver.scala ✓
GuiDriver.scala ✓
fix GUIDriver imports? ✓

TransportationTableau.scala ✓
TransportationTableauSpec.scala ✓

* Compiled v0.3 assembly. Need to upload to github. Also need to do a tutorial-style User Guide. } 28/9.

28/9

- Made screenshots for the User Guide.

Released v0.3 to github!

19/11 — After exams. Finished Example 9 tests + uploaded; updated the readme with future improvements. Also added user guide stuff