

search algorithm

research	5
class design	4
bow line generation	2
area check & length check	1
length check	1
bow point generation	1
min/max width/height determination for bow point	2
bow line symmetry check	2
search sequencing	3
initial investigation (already done)	2
	23

23 +100%
(46 M)

database

initial investigation (already done)	3
download JDBC driver & try using it	4
download trial version of Paradox & try it	2
JDBC familiarization	2
boat lookup	2
	13

13 +25%
(16 M)

logging

2

2 +20%
(2 M)

GUI

boat entry dialog	5
queue display/editing	4
graphical display of lock/boats	10
log request/display	4
menus & general organisation	5
	28

28 +50%
(42 M)

other

system testing	10
visits to Marina	4
customer spec changes/additions	10
user documentation	4
proposal & quote (done)	2
license agreement (done)	3
	33

33 +20%
99 (40 M)
146 M

= 20 weeks
= 5 months.
(= 29 weeks
= 7.2 months)
M

100 days @ £300 = £30000 (£40/hour)
100 days @ £200 = £20000 (£27/hour)
100 days @ £150 = £15000 (£20/hour)

exact algorithm

2	2
4	4
5	5
1	1
1	1
1	1
2	2
1	1
2	2
5	5
25	25

total
(14.4) 5.2

heuristic

3	3
4	4
5	5
5	5
25	25
5	5
15	15

total
(14.4) 13

greedy

2

total
(14.4) 5

CU

2	2
4	4
10	10
4	4
2	2
28	28

total
(14.4) 28

other

10	10
4	4
10	10
4	4
2	2
3	3
38	38

total
(14.4) 38

= 20 units

= 20 units

total
(14.4) 38

total
(14.4) 38

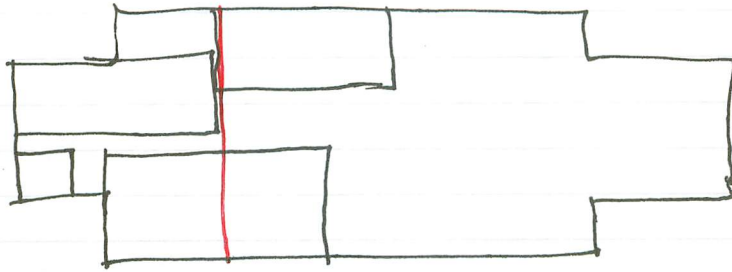
100 kg @ \$300 = \$30,000
100 kg @ \$250 = \$25,000
100 kg @ \$200 = \$20,000

lock packing

~~it should be possible to~~

- P1 • proposition: the lock can be filled in "bow order". i.e. ~~as each boat~~ as each boat is positioned in the lock, its bow will not be further forward than ~~any boat~~ the bow of any boat that is already in the lock. [see CE1.1]

if this proposition is correct, then any space ahead of the bow of the last boat entered is "dead space" - i.e. we do not need to consider putting boats into it.

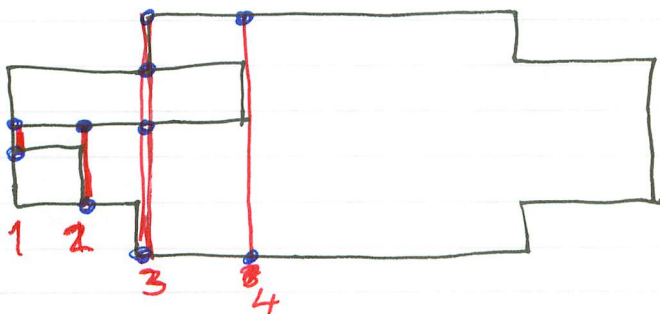


e.g. space to the left of the red line is dead (assuming we are filling from left to right).

- P2 • proposition: the bow of each boat will be positioned against:
- (a) the exit lock gate.
 - or (b) a step in the lock side,
 - or (c) the ~~bow~~^{stem} of another boat.

- P3 • proposition: one side of each boat will be positioned against:
- (a) the lock side,
 - or (b) a side of another boat.

- P4 • proposition: we can consider possible bow positions in left-to-right order.
e.g.:

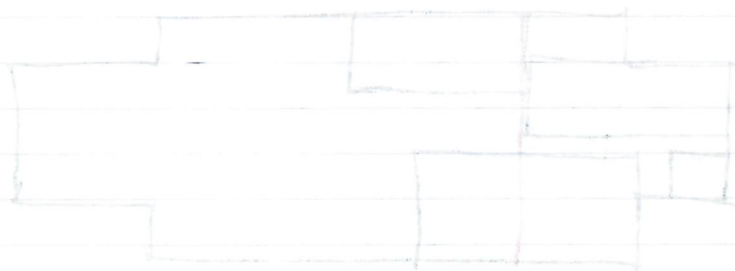


we try all possible boats at line 1. if none fit, we move to line 2 (and ~~consider~~ space to the left of line 2 becomes dead), and so on. note that the red lines are determined by P2. call these red lines "bow lines".

back position

91. Proposition: the back can be filled - "back side" i.e. ~~the back~~ as each back is positioned in the back, its back will not be further forward than ~~any back~~ the back of any back that is already in the back. [see 91.1]

if this proposition is correct, then any space ahead of the back of the last back entered is "back space" - i.e. we do not need to consider putting backs into it.

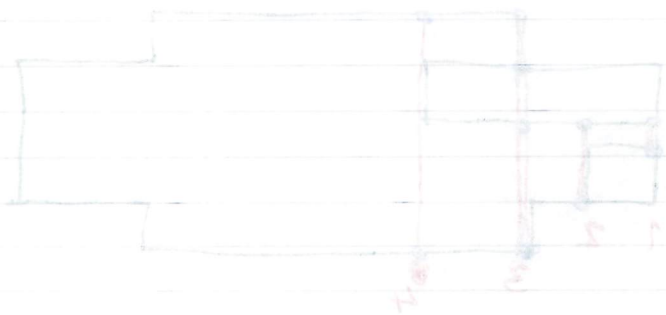


e.g. space to the left of the back line is back (assuming we are filling from left to right).

92. Proposition: the back of each back will be positioned against:
 (a) the next back out.
 or (b) a step in the back side.
 or (c) the back of another back.

93. Proposition: one side of each back will be positioned against:
 (a) the back side.
 or (b) a side of another back.

94. Proposition: we can consider possible back positions in left-to-right order.



we try all possible backs at line 1. if none fit, we move to line 2 (and consider space to the left of line 2 because ahead), and so on. Note that the red lines are obtained by 92. all these red lines "back lines"

Lock packing

P5. proposition: each bow line has ^{points} ~~positions~~ on it where the left or right side of a bow can be positioned, and we need consider only ^{these positions}.
e.g. marked in blue on diagram above.

these points are determined by P3. i.e. we need to find intersections of the bow line with boat sides and lock sides, but only include intersections with boat sides where the boat side extends to the right of the bow line. call these blue points "bow points". [see EP5.1]

P6. proposition: each bow point has a maximum width of boat that can be positioned at it.

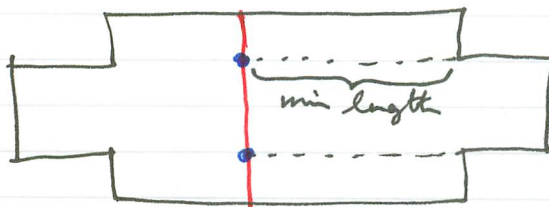
P7. proposition: a bow point may have a minimum width of boat that can be positioned at it.
e.g.:



The minimum width is a consequence of P2.

P8. proposition: a bow point has a maximum length of boat that can be positioned at it.
this is determined by the entrance lock gate or a step in the lock side, whichever is in line with the point.

EP5.1 the steps in the lock sides near the entrance lock gate introduce two additional bow points that need to be considered for large boats:

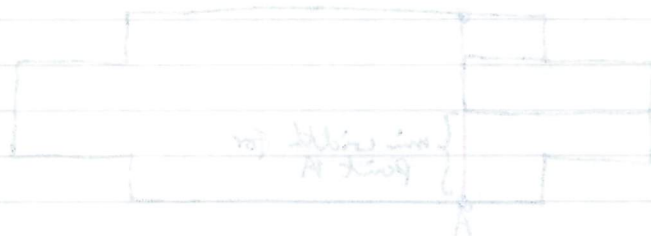


P9. proposition: a bow point may have a minimum length of boat that can be positioned at it.
e.g. see diagram above.

Back bounding

97. Proposition: each bar has its position on it where the left or right side of a bar can be positioned, and we need consider the position of a bar in a diagram where:
- these points are determined by 98, i.e. we need to find intersections of the bar line with back sides and look sides, but only include intersections with back sides where the back side extends to the right of the bar line. All these blue points "bar points". [see 98.2.1]
98. Proposition: each bar point has a minimum width of bar that can be positioned at it.

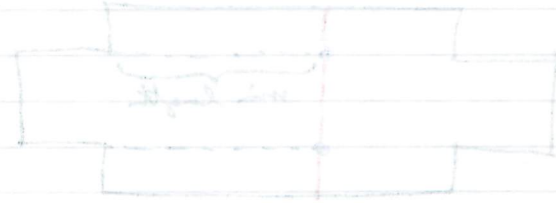
98. Proposition: a bar point may have a minimum width of bar that can be positioned at it.
- e.g.:



the minimum width is a consequence of 95.

98. Proposition: a bar point has a maximum length of bar that can be positioned at it.
- this is determined by the distance back side or a step in the back side, whichever is in line with the point.

- 98.2.1 the steps in the back sides are the distance back side intersections for additional bar points that need to be considered for large bars:

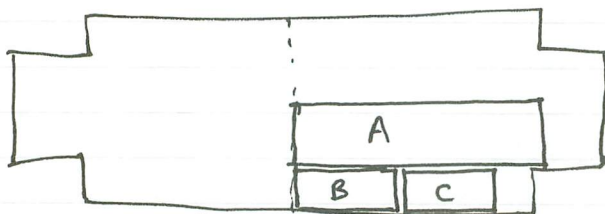


98. Proposition: a bar point may have a minimum length of bar that can be positioned at it.
- e.g. see diagram above.

lock packing

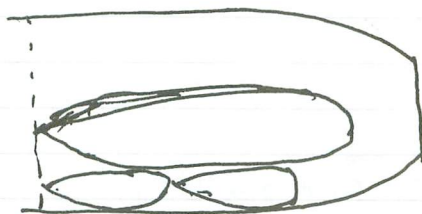
P10. proposition: if a boat satisfies the min and max width and length requirements for a bow point, then it will fit at that bow point. i.e. no further 'fit' tests need to be done.

CE1.1. a boat may restrict access if its stern is in the narrow part of the lock near the entrance lock gate:



the "bow order" of filling would put the boats in the order ABC or BAC, but ~~with this order~~ boat C (and boat B in the ABC order) would not be able to get in past boat A. ~~there~~ ^{two} possible ways of dealing with this problem:

1. abandon "bow order" as a way of ordering the search, and try to find some other way of organizing the search — I'm very reluctant to do this, since the left-to-right progression of bow lines seems to have significant advantages.
2. use "bow order" for ordering the search, but once an organization has been found use a separate algorithm to find a fill order.
3. assume that in practice boats B and C would be able to get past boat A, because the lock and the boats are curved, not rectangles:



4. don't attempt to generate a fill order — the spec doesn't actually ask for it — and let the user determine it.

P11. proposition: as each boat is added it will generate at most one new bow line.

the ~~new~~ new bow line will be at the stern of the boat, and will new if it does not coincide with the stern of another boat.

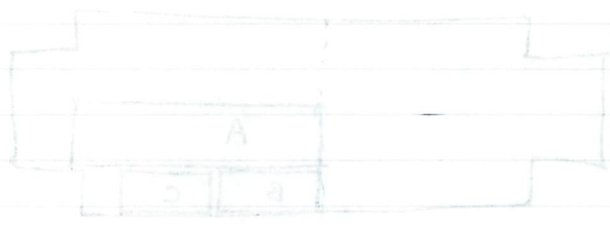
P12. proposition: as each boat is added it will generate at most one new bow point on the current bow line.

the bow point will be ~~at the~~ at the opposite side of the bow from the the bow point to which the boat was added.

book packing

910. Proposition: if a book satisfies the min and max width and length requirements for a book pack, then it will fit at that book pack. i.e. no further 'fit' test need to be done.

911.1. a book may not access if its spine is in the corner part of the book and the outside book parts:



the "book order" of filling would put the books in the order ABC or BAC, but ~~the order~~ book C (and part B in the ABC order) would not be able to get in front part A. This is a possible way of dealing with the problem:

1. absolute "book order" as a way of ordering the count, and try to find some other way of organizing the count - I'm very reluctant to do this, since the left-to-right progression of book lines seems to have significant advantages.
2. use "book order" for ordering the count, but once an organization has been found use a separate algorithm to find a fill order.
3. assume that in practice books B and C would be able to get past book A, because the book and the books are curved, not rectangular:



4. don't attempt to generate a fill order - the space doesn't naturally ask for it - and let the user determine it.

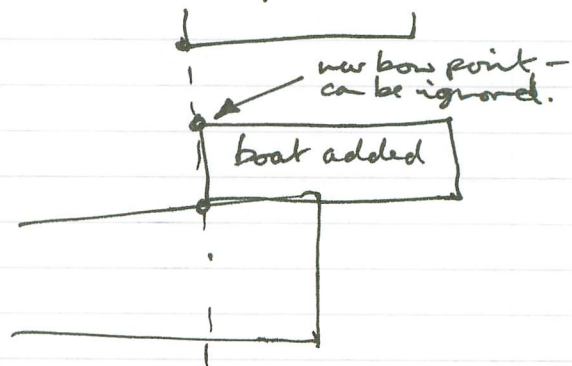
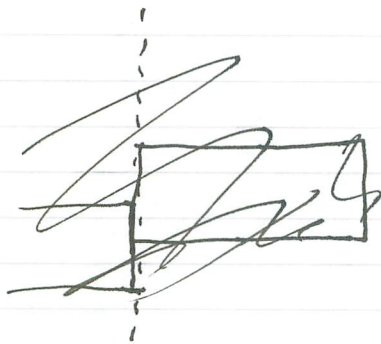
911. Proposition: as each book is added it will generate at most one new book line. The new book line will be at the start of the book, and will have if it does not coincide with the start of another book.

912. Proposition: as each book is added it will generate at most one new book line on the count book line. The book pack will be better at the opposite side of the pack from the book pack to which the book was added.

lock packing

P13. proposition: the bow points on a bow line can be tried in top to bottom order.

i.e. when we add a boat to a bow point we do not need to reconsider any unused bow points above that bow point. also, if adding a boat to a bow point generates a new bow point above that bow point then we do not need to consider that new bow point:



P14. proposition: the bow points on a bow line occur in pairs, corresponding to positioning a boat at the left (bottom) or right (top) of a space.

P15. proposition: we can reject a partial solution if the area left is less than the total area of the remaining boats.

the area left is the area to the right of the current bow line less the area of any boats or parts of boats to the right of the bow line.

P16. proposition: we can reject a partial solution if the ~~area~~ distance from the current bow line to the entrance lock gate is less than the length of a remaining boat.

P17. proposition: the area check suggested by proposition P15 only needs to be applied when we move to a new bow line, adding a boat does not affect the check, since the area left and the total area of the remaining boats is reduced by the same amount.

P18. proposition: considering boats in ~~decreasing~~ decreasing order of their area will reduce the search space, since the area check will fail earlier. ~~(note that the length check of P16 copes fairly well with the most obvious examples where this heuristic might help).~~ (note that the length check of P16 copes fairly well with the most obvious examples where this heuristic might help).

P19. proposition: if a bow line and its set of bow points is symmetrical, then we need consider only half the bow points. actually, the test of symmetry will also need to take into account ~~boats~~ parts of boats that extend to the right of the bow line. this rule can only be applied when we first move to a new bow line.

Book Packing

913. Proposition: the bar points on a bar line can be treated in top to bottom order.

i.e. when we add a book to a bar point we do not need to reconsider any unused bar points above that bar point. Also, if adding a book to a bar point generates a new bar point above that bar point then we do not need to consider that new bar point.



914. Proposition: the bar points on a bar line occur in pairs, corresponding to partitioning a book of the left (bottom) or right (top) of a space.

915. Proposition: we can reject a partial solution if the over left is less than the total area of the remaining books. The over left is the area to the right of the current bar line less the area of any books or parts of books to the right of the bar line.

916. Proposition: we can reject a partial solution if the over distance from the current bar line to the extreme left edge is less than the length of a remaining book.

917. Proposition: the over check suggested by Proposition 915 only needs to be applied when we move to a new bar line, adding a book does not affect the check, since the over left and the total area of the remaining books is reduced by the same amount.

918. Proposition: considering books in decreasing order of their over will reduce the search space, since the over check will fail earlier (left edge of the book is further to the right). (Note that the length check of 916 does fail earlier with the most obvious examples where this length check fails).

919. Proposition: if a bar line and its set of bar points is symmetrical, then we need consider only half the bar points. Actually, the task of symmetry will also need to take into account parts of books that extend to the right of the bar line. This rule can only be applied when we first move to a new bar line.

lock packing.

size of search space:

N	$N!$	3^N	$N! \cdot 3^N$
2	2	9	18
3	6	27	162
4	24	81	2,500
5	120	243	30,000
6	720	729	500,000
7	5040	2241	10,000,000
8	40320	6723	300,000,000
9	362,880	20,169	7,000,000,000
10	3,628,800	60,507	200,000,000,000
11	36,916,800	181,521	6,000,000,000,000
12	413,001,600	544,563	200,000,000,000,000

at each bow point, we have a choice of using that bow point or not, and, if we choose to use it, we have a choice of which boat to put there. many of the bow points will not allow any boat to be positioned there, and so will not add to the search space. a bow point will only add to the search space if it is possible to put a boat there and we also have to consider not putting a boat there.

in most cases, all the boats that can be positioned at the top bow point of a pair can also be positioned at the bottom point of the pair.

hopefully, whenever we have tried all (or most) of the boats at a pair of bow points, the search space arising from not putting a boat at the bow point pair will be severely pruned on grounds of area.

so, optimistically, we would only have 2 bow points to consider at each stage in the search - but let's say 3 bow points, to be more realistic.

a 1GHz CPU can perform roughly 1,000,000,000 instructions per second.

a step of the search is likely to need at least 1000 instructions - let's say 10,000 to be on the safe side. so we can perform roughly 100,000 search steps per second. this suggests that $N=7$ (100 seconds) would be the limit for exhaustive searching.

back looking.

size of search space:

n	$U!$	2^n	$N!$
2	2	4	18
3	6	27	182
4	24	81	2,000
5	120	243	300,000
6	720	729	20,000,000
7	5,040	2,187	10,000,000,000
8	40,320	6,561	300,000,000,000
9	362,880	19,683	70,000,000,000,000
10	3,628,800	60,000	200,000,000,000,000,000
11	39,916,800	1,771,471	6,000,000,000,000,000,000
12	479,001,600	16,777,216	200,000,000,000,000,000,000

at each point, we have a choice of using that point or not, and, if we choose to use it, we have a choice of which point to put there. many of the points will not allow any point to be positioned there, and so will not add to the search space. a point will only add to the search space if it is possible to put a point there and we also have to consider not putting a point there. in worst case, all the points that can be positioned at the top are part of a pair can also be positioned at the bottom part of the pair. hopefully, however we have tried all (or most) of the points at a pair of points, the search space arising from not putting a point at the last point pair will be severely pruned on grounds of size. so, optimistically, we would only have 5 points to consider at each step stage in the search - but let's say 3 points to be more realistic.

a little CPU can perform roughly 1,000,000,000 instructions per second. a step of the search is likely to need at least 1000 instructions - let's say 10,000 to be on the safe side, so we can perform roughly 100,000 steps per second. this suggests that 1M (100 seconds) would be the limit for exhaustive searching.