CS404 Assignment Report

Matthew Wight (2008398)

Basic Strategies

Before testing more sophisticated approaches, I tested various `flat_bot` scripts against each other:

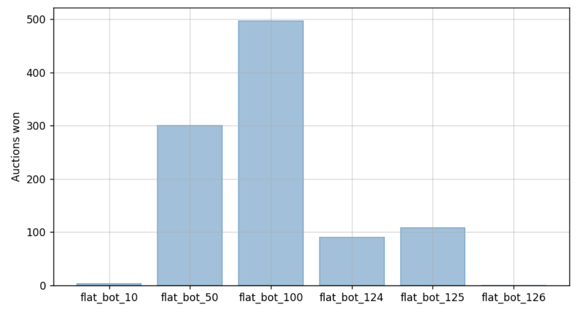

Fig. 1: Success of `flat_bot` variants in auctions (1,000 runs).

Mathematically, `flat_bot_126` (or any `flat_bot` bidding more than 125) can never win any game because a bot needs to win at least eight rounds to win a game, and $126 > \frac{1001}{8}$, so the bot will always run out of money before winning the game.

By inspecting the individual auctions, I found that `flat_bot_125` always lost the game if it bought four of any one painting type, or two of any three painting types. This is because one or more of the paintings it bought were surplus to the game's winning conditions, causing the bot to run out of money before it acquired the eight paintings it would need to win. The `flat_bot_100` variant was more successful in testing because it was able to buy up to two redundant paintings before running out of money.

We can easily fix this issue in our strategy by checking our current collection and the current painting type being sold, submitting a bid only if the current painting is not redundant in our collection. I implemented this as `basic_bot` and added it to the previous room of `flat_bot` variants:

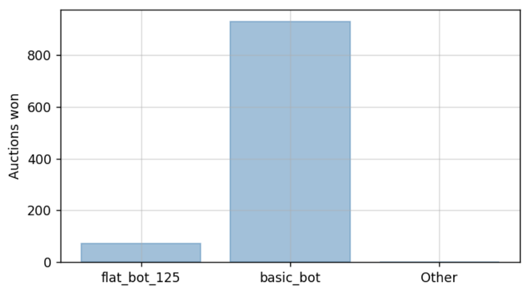

Fig. 2: Performance of `basic_bot` against `flat_bot` variants over 1,000 runs.

The `basic_bot` is able to successfully outbid every `flat_bot` bidding less than 125 without running out of money.

I iterated on `basic_bot` to create `prop_bot`. Where the former version would always submit a bid of 125, `prop_bot` instead submits the bid $\frac{\text{remaining budget}}{\text{min \# paintings to win}}$. This is functionally highly similar to `basic_bot`, but has the advantage of bidding 126 in the final round. It also has the ability to adapt to the changes we will make to this bot in the future (because it bids proportionally to its remaining budget instead of a flat amount).

Testing

Some of the differences between the strategies are based on optimising decision making in edge cases, meaning the overall win rate will only differ by a small amount. This makes it especially important to be sure that an improvement in empirical win rate of one bot over another is due to a superior strategy rather than just random chance. For this reason, I have chosen to list confidence intervals for empirical testing instead of overall win rate.

I tested the different bot strategies against a variety of opposition. For each combination of opponents, I ran 1,000 to 2,000 auctions on each room and used a binomial test to calculate a confidence interval for the bot's win rate at a confidence level of 95%. This allows me to estimate which bots are most successful in each room to a high degree of confidence(1).

Avoiding Losses

After observing many auctions between the primitive bots, I found that `prop_bot` often lost games despite having a substantial remaining budget to use. It often bid 0 or 125 on the final round, losing to another bot which would bid 125 (or 126) to buy the winning painting. The next bot, `mean_bot`, aims to outbid opponents who are able to win the game within the next round.

First, let us say that a bot is *likely-to-bid* if the bot cannot win the auction without at least winning a painting of the same type as is being sold in the current round. For example, if a bot's current collection is $[2R, 1V, 1P, 1D]$ and the current painting is a Picasso, then the bot is likely-to-bid.

Next, we can define *almost-winning* like so: a bot is almost-winning if it is able to win the game by winning the next round. We can determine this attribute for each bot by checking their painting collection and the current painting being sold.

Using this information, `mean_bot` is able to gain an advantage by outbidding (submitting a bid one greater than) the highest possible bid of any almost-winning player[2]. The bot will attempt to outbid whether or not it would benefit from having the painting in its collection.

Opposition	Lower bound	Upper bound
`prop_bot`	0.594	0.655
9 × `flat_bot_125`	0.557	0.619
4 × `prop_bot`	0.203	0.255
9 × `prop_bot`	0.087	0.126

TABLE I: Empirical win rate of `mean_bot` (confidence level 95%).

The test results above show that `mean_bot` performs well against `prop_bot`, winning one-on-one in a majority of games. The new approach also beats a room of nine `flat_bot_125` more often than not.

Another idea to consider is that it could be beneficial to let a player who isn't almost-winning outbid the almost-winning player. This is because outbidding the almost-winning players will lower our remaining budget, making it less likely that we can win subsequent rounds—especially if we were forced to bid a large amount, giving other players an advantage for free[3]. However, since there is no way of knowing how much other players will bid, this would be detrimental in many cases.

I implemented this idea by checking whether any of the likely-to-bid players had enough remaining budget to outbid all of the almost-winning players. In testing, I found this to decrease `mean_bot`'s win rate. To improve on this idea, I implemented the following decision process:

```
$M, M^* \leftarrow \emptyset$
for all likely-to-bid bots $b$ do
    $n \leftarrow$ min. # paintings needed for $b$ to win
    $M \leftarrow M \cup \{b.\texttt{budget}//n\}$
end for
for all almost-winning bots $b$ do
    $M^* \leftarrow M^* \cup \{b.\texttt{budget}\}$
end for
if $\max(M) \leq \max(M^*)$ then
    Attempt to outbid.
else
    Follow usual mean_bot procedure.
end if
```

Using this process, the `mean_bot` had marginally improved test performance to the original iteration. Additionally, I believe this version to be more robust to sophisticated strategies.

Opposition	Lower bound	Upper bound
`prop_bot`	0.593	0.654
9 × `flat_bot_125`	0.567	0.629
4 × `prop_bot`	0.218	0.272
9 × `prop_bot`	0.082	0.120

TABLE II: Win rate of the new `mean_bot`.

CPM and Slack

The next strategy I tested is inspired by the critical path method (CPM) used in project management[4]. By looking at our current painting collection and the upcoming sequence of paintings in the auction, we can calculate the smallest number of rounds it would take to win the game. This is done with a simple iterative algorithm:

```
$collection \leftarrow$ our painting collection so far
$T \leftarrow [3, 3, 1, 1]$
for $i$ in remaining rounds do
    increment $collection[i^{th}$painting sold$]$
    $C \leftarrow \text{sort}(collection.values)$
    if $\min(C - T) \geq 0$ then
        return $i$
    end if
end for
```

We can also inspect the array $(C - T)$ to see how many redundant paintings of each type appear in the minimum i rounds it will take to win the game (i.e. the *slack* of the painting type). We can also think of slack in terms of number of rounds. For example, given current collection $[4R, 1P, 1V, 0D]$ and upcoming painting sequence $(P, V, D, D, V, P, R, ...)$, we can calculate the slack for each painting:

Slack type	Picasso	Van Gogh	Da Vinci
Paintings	1	0	1
Rounds	4	0	2

TABLE III: Slack for each painting in example.

We can also draw a critical path diagram[5]:

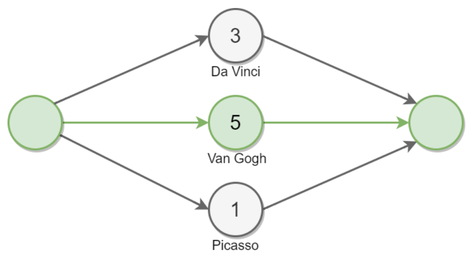

Fig. 3: CPM analysis of example (in rounds).

This shows that in this example, Van Gogh is 'critical', whereas Picasso and Da Vinci are subcritical. In project management, the aim is to prioritise completion of the critical path in order to minimise the length of the project as a whole[4]. We can apply the same concept to the auction by submitting higher bids for critical paintings or lower bids for subcritical paintings. The intuition behind this strategy is that it should increase the likelihood of collecting the required eight paintings in the shortest time possible.

I tested several approaches to adjusting bids based on the schedule criticality of each painting and found a simple formula that appeared to be effective:

$$\text{bid}_i \leftarrow \text{bid}_i - \text{painting slack}_i$$

I then tested the new approach in auctions against a variety of bots:

Opposition	Lower bound	Upper bound
`prop_bot`	0.632	0.691
9 × `flat_bot_125`	0.550	0.612
`mean_bot`	0.537	0.599
4 × `mean_bot`	0.307	0.366
2 × `mean_bot`, 2 × `prop_bot`	0.187	0.239
4 × `prop_bot`	0.173	0.223
9 × `prop_bot`	0.088	0.115

TABLE IV: Empirical win rate of `slack_bot`.

We can see that `slack_bot` outperforms previous bots one-on-one, showing that the slack strategy helps the bot to win in some cases. However, the improved performance disappears against a larger number of `prop_bot` opponents. This could be because `prop_bot` always bids at least 125 for non-extraneous paintings, and as a result is more likely to win a painting on our player's 'critical path'. The same is not true for `mean_bot` because its proportional bid can be below 125 in some cases.

PERT

After the success of the CPM slack-adjusted bidding approach, I looked to expand on this idea using similar principles from project management. The Program Evaluation and Review Technique (PERT) is an extension to CPM, allowing project managers to calculate an expected task duration based on three estimates (best case, worst case and most-likely case)[6].

The best and worst case estimates are calculated by looking at the upcoming sequence of paintings to be sold and the number of likely-to-bid bots for the painting type. Finding the most-likely case estimate follows a more complex process: the likely-to-bid bots are ordered based on their remaining budget and the number of paintings they have left to collect, then `pert_bot`'s most-likely estimate is determined by its position in this ordering. These estimates are incorporated into the PERT formula, which is an approximation of the beta distribution formula:

$$\text{PERT}_{i,p} = \frac{1}{6}(\text{best}_{i,p} + \text{worst}_{i,p} + 4 \cdot \text{most-likely}_{i,p})$$

At the start of each round i, $\text{PERT}_{i,p}$ is calculated for each painting type p. The PERT slack for painting p being sold in round i is given by:

$$\text{PERT slack}_i = \max_q(\text{PERT}_{i,q}) - \text{PERT}_{i,p}$$

This PERT slack formula is intended to provide a more realistic slack estimate than the formula used in `slack_bot`.

An issue that became apparent in testing was that `pert_bot` underperformed slightly against many `prop_bot` opponents. I addressed this by incorporating the size of the room into the bid adjustment formula:

$$\text{bid}_i \leftarrow \text{bid}_i - (\text{PERT slack}_i) \cdot (1 - \frac{\text{\# opponents}}{10})$$

Testing showed further improved results for `pert_bot` over `slack_bot`:

Opposition	Lower bound	Upper bound
`flat_bot_125`	0.996	1.000
`prop_bot`	0.756	0.808
`mean_bot`	0.602	0.663
9 × `flat_bot_125`	0.569	0.631
`slack_bot`	0.534	0.596
4 × `prop_bot`	0.179	0.230
9 × `prop_bot`	0.0730	0.109

TABLE V: Empirical win rate of `pert_bot`.

Overall, the strongest bot based on my tests was `pert_bot`. This is the bot I chose to include in my submission.

References

[1] J. W. Fraas and I. Newman, "A binomial test of model fit," *Structural Equation Modeling: A Multidisciplinary Journal*, vol. 1, pp. 268–273, 01 1994.

[2] S. T. Anderson, D. Friedman, and R. Oprea, "Preemption games: Theory and experiment," *American Economic Review*, vol. 100, pp. 1778–1803, 09 2010.

[3] T. R. Palfrey and H. Rosenthal, *Testing Game-theoretic Models of Free Riding*. 1990.

[4] J. A. Carruthers and A. Battersby, "Advances in critical path methods," *Journal of the Operational Research Society*, vol. 17, pp. 359–380, 12 1966.

[5] J. Santiago and D. Magallon, "Critical path method," 2009.

[6] H. O. Hartley and A. W. Wortham, "A statistical theory for pert critical path analysis," *Management Science*, vol. 12, pp. B–469–B–481, 06 1966.