Offline Algorithms in Low-Frequency Trading

CLEARING COMBINATORIAL AUCTIONS

TERENCE KELLY

MONEY TALKS

Expectations run high for software that makes real-world decisions, particularly when money hangs in the balance. This edition of Drill Bits shows how well-designed software can effectively *create wealth* by finding subtle opportunities for gains from trade. We'll unveil a deep connection between auctions and a classic problem from our school days, we'll see that clearing an auction—allocating resources based on bids—resembles a high-stakes mutant Tetris game, and we'll learn to stop worrying and love an NP-hard problem that's far from intractable in practice.

My colleagues and I developed the techniques of this column for two contexts: rental markets for computational resources in what was once quaintly called *utility computing* (later renamed *grid computing*, then the *cloud*);^{4,5} and procurement auctions whereby manufacturers acquire physical parts.² To steer clear of details that I'm not at liberty to discuss, this column uses a different example problem, but the underlying principles transcend specific applications, and the differences are superficial.

Indeed, the key to writing efficient software that ferrets out every last dollar is to see beyond the distracting particulars of a real-world problem and map it onto a well-known formal problem. Once we make the

right connection, elegant code almost writes itself. The software that accompanies this installment of Drill Bits includes succinct code that clears auctions.

OFFICE SPACE

Consider an office building whose owner leases its rooms and parking spaces to firms of different sizes. Prospective tenants differ in their *absolute* resource needs, as well as in the *ratio* of rooms to parking spaces they require: some tenant firms' employees drive to work solo; other occupants carpool or ride public transit. Some tenants give their employees private rooms; others cram several workers into each room. Demand varies with expense: tenants make do with less if resources are dear but grab more if resources are cheap. Most importantly, rooms and parking spaces complement one another such that the usefulness of either depends on how many of the other are acquired. If the two resources were allocated independently, tenants might end up with lopsided combinations. Eliminating this risk requires unified allocation of both rooms and parking spaces.

The straightforward approach is to use an auction as follows: Prospective tenants submit bids consisting of one or more resource *bundles* labeled with offer prices. For example, a bid might say, "I'd pay \$18K for exactly 3 rooms and exactly 6 parking spaces; or \$26K for 4 rooms and 8 spaces; or \$31K for 5 rooms and 12 spaces." Every bid contains an implicit "zero bundle" that offers \$0 for no resources. Every prospective tenant firm will receive a resource bundle from its bid—possibly the zero bundle—and pay the bundle's offer price.

1

Figure 1 illustrates resource bundles and bids in our two-resource office example. Figure 1a represents a resource bundle as a rectangle of height and width proportional to the number of rooms and parking spaces in the bundle. Figure 1b depicts three bids distinguished by color. Rectangle shapes—wide, tall, square—reflect the different ratios of rooms to parking spaces that suit different tenants.

Bids may contain any number of bundles. Longer bids force the bidder to reckon offer prices for a greater number of resource bundles; the benefit is that longer bids are required to express a willingness to snap up bargains opportunistically. The building owner can forbid unacceptably low bids by imposing price floors (e.g., derived from resource costs).

FIGURE 1: **VISUALIZING RESOURCE BUNDLES & BIDS**

[a] Resource bundle

[b] Bids (zero bundles not shown)

Clearing our auction means selecting exactly one bundle from each bid without allocating more resources than are available. Figure 2 depicts a feasible allocation: Colored rectangles representing bundles selected from bids fit corner to corner within the outer dashed rectangle representing the numbers of available rooms and parking spaces. Among all feasible ways of clearing an auction, we seek an *optimal* solution that maximizes revenue (i.e., the sum of the selected bundles' offer prices), given the bids that were submitted. An optimal solution need not allocate all available resources, and different bidders might wind up paying different prices for similar or identical resource bundles.

Our example office-space auction is known in the research literature as a “multi-unit combinatorial auction

FIGURE 2: **CLEARING THE AUCTION: A FEASIBLE ALLOCATION**

2

auction haiku

*clearing an auction
is like packing a knapsack
choose wisely, get rich*

with exclusive-OR bids”: *combinatorial* because it allocates several resource types simultaneously; *multi-unit* because identical copies of each resource are available; and *XOR* because each bid lists several resource bundles, of which exactly one is awarded to the bidder.

LIFTING THE VEIL

The right analogy links the practical problem of clearing our example auction to a well-known formal problem:

Allocating a bundle of resources to a bidder reduces the pool of available resources in precisely the same way that placing a physical object in a container reduces the container’s capacity to hold additional objects. Clearing our auction is a generalization of the classic knapsack problem.

The auction-knapsack connection is clearest for a simple auction that allocates a single type of resource—say, rooms in an office building—across bids, each containing a single quantity/price pair. The number of rooms in the building corresponds to the weight capacity of the knapsack; a bid for a specified number of rooms at a stated offer price corresponds to a candidate object with a weight and “profit”; and optimally clearing this simple auction corresponds to solving the classic *binary* knapsack problem, so named because each object is either chosen or not, by stuffing the most profitable subset of objects into the knapsack without exceeding its weight limit. Given this correspondence between the problem of clearing a simple auction and the binary knapsack problem, generalizing both problems in tandem yields fairly comprehensive *taxonomies* of auction-clearing problems and knapsack

problems that align precisely, differing only in the interpretation of their variables.⁵

The problem of optimally clearing our example office-space auction corresponds to the “multidimensional multiple-choice knapsack problem”:^{4,5} The knapsack now has two aspects of capacity, weight and volume (corresponding to rooms and parking spaces); each candidate object (priced resource bundle) has a weight, volume, and profit; objects are grouped into categories (bids); and one object is chosen from each category to maximize profit without exceeding either aspect of the knapsack’s capacity. Though less famous than the classic binary knapsack problem, this generalization is known.³

Mapping a practical auction-clearing problem onto a well-studied knapsack problem brings important benefits. Most importantly, making this connection enables us to exploit the wide range of excellent solution methods that computer scientists and operations researchers have developed since the dawn of the computer era. We’ll drill down on the most elegant solution, which works remarkably well in practice.

PACKING TENANTS PROFITABLY

The auction-knapsack connection cautions against hacking together ad hoc clearing algorithms, which fail to maximize revenue. It’s possible, for example, to clear our auction with a “greedy” heuristic that selects bundles with the greatest ratio of offer price to bundle “size.” Unfortunately, the knapsack literature shows that greedy heuristics may leave a pile of money on the table.³

At the opposite end of the solution-quality spectrum

3

is the brute-force approach of choosing a revenue-maximizing feasible solution among all possible ways of selecting one bundle from each bid. Brute force would find all possible gains from trade, including those overlooked by suboptimal heuristics, but it requires $O(B^T)$ time for T tenant bids, each containing B resource bundles.

The challenge of clearing our example auction both optimally and efficiently appears daunting because it seems to involve *simultaneous* interdependent decisions about *all* submitted bids. One way to tame the problem is to decompose it into a *sequence* of decisions about *individual* bids. Consider the four possible outcomes for bid 1 from figure 1b, shown in figure 3: A solution must include one of the three nonzero bundles or the zero bundle. Deciding what to do with bid 1 leaves a reduced pool of remaining resources and one fewer bid to worry about—a *smaller version of the original auction*. There are four ways to choose a bundle from bid 1, then optimally

FIGURE 3: ALTERNATIVES FOR BID 1

solve the smaller “inner” clearing problem that remains; the best of these four alternatives optimally solves the original “outer” problem.

Because all optimal solutions have this “nested” structure, standard dynamic programming can find an optimal solution. Our two-resource example problem is a bit more complicated than typical dynamic programming exercises in undergrad computer science curricula, but the basic idea is the same. At center stage is the definition of optimal revenue as a recursive function of *subsets* of submitted bids and available resources: If M rooms and S parking spaces are available to be auctioned across bids numbered 1 through T , the optimal revenue from auctioning $m \leq M$ rooms and $s \leq S$ spaces across the first $t \leq T$ bids is $F[t, m, s]$, shown in figure 4, where the max on the third line is taken over all bundles b in bid t , and b_s , b_m , and b_s , respectively, denote the offer price, number of rooms, and number of parking spaces of bundle b . The definition of F simply formalizes common sense: The first line forbids infeasible allocations; the second line says that revenue

4

FIGURE 4: OPTIMAL REVENUE

$$F(t, m, s) = \begin{cases} -\infty & \text{if } m < 0 \text{ or } s < 0 \\ 0 & \text{if } m \geq 0, s \geq 0, \text{ and } t = 0 \\ \max_{b \in t} \{ b_s + F(t-1, m-b_m, s-b_s) \} & \text{if } m \geq 0, s \geq 0, \text{ and } 1 \leq t \leq T \end{cases}$$

is zero if there are no bids; the third line defines “outer” optima in terms of “inner” optima.

Evaluating $F(T, M, S)$ yields the maximum revenue available from the given “outermost” auction. An implementation must keep track of the decisions that led to the optimal solution [i.e., the bundles selected by the max operation on the third line of the definition of F]. An implementation should also cache previously computed F values to avoid redundant work. My auction-clearing program uses multidimensional arrays for these purposes.

Clearing our auction via dynamic programming requires at most $O(BTMS)$ time and $O(TMS)$ memory.⁵ Technically, these bounds are *pseudo*-polynomial [i.e., polynomial in problem-size parameters M and S but not in the length of a problem instance]. The classic binary knapsack problem is NP-hard, as is the generalization corresponding to clearing our office-space auction, so don’t hold your breath for a truly polynomial solution. Fortunately, knapsack problems admit pseudo-polynomial algorithms that are fast enough in many real-world applications, thus debunking the widespread overgeneralization that “NP-hard problems are intractable.”¹⁰ Learn to recognize knapsack problems in the wild, approach them with cautious optimism, and consider simple solutions based on dynamic programming.

DRILLING DEEPER

Unless countermeasures are taken, bidders in auctions may seek advantage by offering prices below the true value they would derive from resources. In a simple single-item auction, for example, if you’re willing to pay \$100 for the item, *and you believe that no one else values it above \$50*,

you would be tempted to offer a mere \$50.01—why pay more? If your beliefs about others are mistaken, however, you might lose—for example, to a rival who offers \$53. The classic remedy for single-item auctions, the famous Vickrey auction, awards the item to the highest bidder at the price offered by the *second-highest* bid. It can be shown that the best strategy in a Vickrey auction is to offer one's true valuation.¹⁴ The Generalized Vickrey Auction uses a more sophisticated repricing trick to create incentive for truthful bidding in *any* kind of auction.⁹

Our example office-space auction includes only two types of resources—rooms and parking spaces—but we could define more (e.g., by distinguishing small, medium, and large rooms as different resources). Unfortunately, the time required to clear the auction via dynamic programming is exponential in the number of resource types (which corresponds to the “dimensionality” of a multidimensional knapsack problem); the same is true, in the worst case, for every known solution method. Compared with most other methods, dynamic programming has the advantage that simple back-of-the-envelope calculations based on asymptotic time and memory demands quickly reveal whether dynamic programming can tackle any specific instance of an auction-clearing problem similar to ours.⁵ If such calculations indicate that dynamic programming can't handle a given problem instance satisfactorily, the next approach to consider is integer programming.¹

General-purpose integer programming has the added benefit of accommodating a wide range of *side constraints* on solutions. In our office-space auction, for example, the

building owner might insist that the number of tenants awarded leases must lie in a specified range. If the owner can't articulate side constraints explicitly but "knows a good solution when she sees one," she can employ an alternative method that generates multiple high-revenue feasible solutions to the auction-clearing problem, from which she may select her favorite.²

High-frequency trading provides several contrasts to "low-frequency trading" via combinatorial auctions. Matching buy and sell orders at a stock exchange¹³ is less computationally intensive than optimally clearing an auction like ours, but high-frequency trading software that places orders at a stock exchange must gather and analyze relevant data, make decisions, and act under realtime pressure.⁸ A more important contrast involves social utility: optimal auction-clearing software effectively transforms computational effort into newly created wealth by finding gains from trade that would be overlooked by suboptimal methods. In contrast, high-frequency trading is frequently castigated for *destroying* wealth.^{7,12}

For a different problem at the intersection of programming and finance, check out the connection between currency arbitrage and graph analysis in *Algorithms* by Sedgewick and Wayne.¹¹ Hone the skill of mapping real-world problems onto formal ones by studying examples such as this.

Bits

Example code for this column is at https://queue.acm.org/downloads/2020/Drill_Bits_O3_sample_code.tar.gz. My

an auction-clearing program is based on dynamic programming; for small problem instances it checks its solution against the brute-force algorithm. My random bid generator can create inputs for the clearing program; I provide a script that runs both programs together.

Drills

Auctions harness the full range of programming skills, from theory to software engineering, for intensely practical purposes. Try a few of the following exercises, listed roughly in ascending order of difficulty.

1. In keeping with the Drill Bits tenet of avoiding needless work,⁶ my auction-clearing program evaluates a dynamic program “top down.” Whereas “bottom-up” evaluation would completely fill the arrays that cache F values and record optimal decisions, top-down evaluation fills only those array entries that contribute to the solution. Are there ever good reasons to use the bottom-up approach?
2. Formulate the clearing problem as an integer program⁵ and feed it to a solver such as CPLEX or Gurobi. Compare with dynamic programming in terms of convenience, flexibility, and performance.
3. In the lingo of knapsack research, my random bid generator creates “uncorrelated” problem instances, which are relatively easy for some solution methods.^{3,5} Modify the code to output harder “strongly correlated” instances. Compare the performance of integer programming and dynamic programming on easy and hard instances.
4. Implement a Generalized Vickrey Auction for the office-space auction.⁹ How much does it increase the

SCIENCE TIP: IF YOU HAVE A COOL CONCEPT YOU NEED A NAME FOR, TRY "DYNAMIC ENTROPY."

<https://lxcdd.com/2318/>

computational cost of clearing?

5. My auction-clearing program supports “forward” auctions. Implement general two-sided exchanges in which bids may offer to buy and sell resources simultaneously.
6. Generalize my auction-clearing program to handle an arbitrary number of resource types. (Hint: Hash tables might be better than arrays for tracking optimal decisions and caching.)
7. Implement a branch-and-bound algorithm for clearing auctions; avoid reinventing wheels.¹ Compare it against integer programming and dynamic programming on hard problem instances.
8. Make a fortune in commercial real estate auctions. Run for president.

Acknowledgments

Jeff MacKie-Mason, professor of economics and university librarian at UC Berkeley, provided valuable feedback on an early draft. Juno Kim of UC San Diego reviewed the example software, squashing a bug or two.

References

1. Andersson, A., Tenhunen, M., Ygge, F. 2000. Integer programming for combinatorial auction winner determination. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)*; <https://doi.ieeecomputersociety.org/10.1109/ICMAS.2000.858429>; <http://www.eecs.harvard.edu/~parkes/lcs286r/spring02/papers/icmas00.pdf>.

2. Byde, A., Kelly, T., Zhou, Y., Tarjan, R. 2009. Efficiently generating k -best solutions to procurement auctions. In *Algorithmic Aspects in Information and Management*, ed. A. V. Goldberg and Y. Zhou. Lecture Notes in Computer Science 5564. Springer; https://doi.org/10.1007/978-3-642-02158-9_8; <http://www.hpl.hp.com/techreports/2009/HPL-2009-163.pdf>. Also U.S. Patents #7,801,769 and #8,086,520.
3. Kellerer, H., Pferschy, U., Pisinger, D. 2004. *Knapsack Problems*. Springer.
4. Kelly, T. 2003. Utility-directed allocation. In *Self-Managing Systems* (June); <https://www.hpl.hp.com/techreports/2003/HPL-2003-115.pdf>. Also U.S. Patent #7,844,967.
5. Kelly, T. 2004. Generalized knapsack solvers for multi-unit combinatorial auctions. In *Agent-Mediated Electronic Commerce VI*, ed. P. Faratin and J. A. Rodríguez-Aguilar. Lecture Notes in Computer Science 3435. Springer; https://doi.org/10.1007/11575726_6; https://web.eecs.umich.edu/~tpkelly/papers/LNAI_3435_0073_auction_knapsack_connection.pdf.
6. Kelly, T. 2020. Efficient graph search. *acmqueue* 18(4); <https://queue.acm.org/detail.cfm?id=3424304>.
7. Krugman, P. 2009. Rewarding bad actors. *New York Times* (August 2); <https://www.nytimes.com/2009/08/03/opinion/03krugman.html>.
8. Loveless, J., Stoikov, S., Waeber, R. 2013. Online algorithms in high-frequency trading. *acmqueue* 11(8); <https://dl.acm.org/doi/10.1145/2523426.2534976>.
9. MacKie-Mason, J. K., Varian, H. R. 1994. *Generalized Vickrey Auctions*. University of Michigan Department

- of Economics Technical Report; <http://hdl.handle.net/2027.42/41250>.
10. Papadimitriou, C. H., Steiglitz, K. 1998. *Combinatorial Optimization: Algorithms and Complexity*, 388. Dover.
 11. Sedgewick, R., Wayne, K. 2011. *Algorithms*, 4th edition, 679. Addison-Wesley.
 12. Stiglitz, J. E. 2014. Tapping the brakes. In *Federal Reserve Bank Atlanta Financial Markets Conference* (April); <https://www.frbatlanta.org/-/media/Documents/news/conferences/2014/fmcl/Stiglitz.pdf>.
 13. Treleaven, P., Galas, M., Lalchand, V. 2013. Algorithmic trading review. *Communications of the ACM* 56(11), 76–85; <https://doi.org/10.1145/2500117>.
 14. Varian, H. R. 2008. Designing the perfect auction. *Communications of the ACM* 51(8), 9–11; <https://cacm.acm.org/magazines/2008/8/5343-designing-the-perfect-auction/fulltext>.

Terence Kelly (tpkelly@acm.org) has devoted years to the theory and practice of auctions and pricing. His work in industry led to several publications, patents, and tech transfers related to auction design, cloud resource allocation, and material parts procurement. Kelly would get rich if only he would choose wisely.

Copyright © 2020 held by owner/author. Publication rights licensed to ACM.