The Importance of Perfect Markets

The practical and performant first step towards combination matching is single-step implications; computing the bait orders involving precisely one combination. Even these are not without technicalities (baits hitting baits, ratios causing volume constraints, etc.), but where the complexity, both conceptual and computational, really skyrockets is when one attempts to extend this to imply across multiple combinations, causing complex volume interdependencies which interact with volume constraints and priorities in complicated ways.

To our knowledge no system exists which achieves advanced bait generation while retaining well-defined formal semantics. It is, however, not very complex to at least formally define a general perfect matcher, one which finds all possible trades and displays all available volume. We offer a sketch of the theoretical basis of such a matcher in Section 1. Section 2 discusses how our product can efficiently implement a broad variety of sound semantics by starting from the perfect case. Such a perfect matcher is useful, or even necessary, in several scenarios.

Implementing Approximate Markets: Non-functional requirements often (appear to) make approximate matching necessary. Even there a perfect matcher serves several key roles.

1. It allows measuring the approximation, determining what fraction of a perfect market is achieved. E.g. verifying that e.g. 80% of the volume in a testing dataset is covered. When replacing one approximate matcher with another this is especially important, as they will often be incomparable (i.e. one approximation will not be strictly better than the other).
2. It makes it possible to define the approximation in formal logical term (see Section 1) to test and validate that the new implementation is correct.
3. The perfect matcher can be used to compute the actually (objectively) computationally difficult cases in a dataset, to fairly compare performance and test on the relevant instances.

For Operators of Existing Markets (and Sophisticated Participants): Monitoring and reporting on how the state of the market compares to a perfect state can instruct both business and technology decisions. For example listing additional combination products to exploit otherwise undiscovered volumes, or simply better trading strategy for participants.

Implementing (Usually-)Perfect Markets: As Section 2 discusses the real-world implementation of perfect matchers is a realistic proposition, and for many (more slow-moving) types of markets this may be the entirely correct tradeoff (in some cases regulatory constraints make complexity necessary). Even more frequently usually-perfect markets are also be desirable, with well-defined and predictable failure modes (i.e. announcing when and how less than full volume is advertised). The technology can also be used to define auctions with implications.

1 Perfect Matching

We loosely define perfect matching in terms of just orders, leaving e.g. orderbooks, tick sizes, leg pricing, etc. implied by the existence of the order. Many details are elided but can easily be included within this framework.

Definition 1: An *order o* has a quantity $\mathrm{qty}(o) \in \mathbb{N}^+$, a priority $\mathrm{prio}(o) \in \mathbb{N}$, a cost $\mathrm{cost}(o) \in \mathbb{Z}$ (a negative cost corresponding to a participant paying to have it executed), and a collection of contracts to be traded (e.g. an order for a single contract is in an outright). A *market $\mathcal{O}$* is a set of orders.

Then a trade is simply any selection of (parts of) orders such that all the contracts trade out, and the sum cost is less than or equal to zero.

Definition 2: For a market $\mathcal{O}$ a *candidate trade t* is a mapping $t : \mathcal{O} \to \mathbb{N}$ (usually $t(o) = 0$ for "most" $o \in \mathcal{O}$) such that

1. $t(o) \leq \mathrm{qty}(o)$ for all $o \in \mathcal{O}$,
2. $0 \geq \sum_{o \in \mathcal{O}} t(o) \cdot \mathrm{cost}(o)$, and,
3. the sum of contracts traded is zero[1].

Let $\sup(t)$ denote the set $\sup(t) = \{o \mid t(o) > 0\}$. Let $\mathcal{T}_{\mathcal{O}}$ denote the set of all candidate trades for the market $\mathcal{O}$.

A perfect market $\mathcal{O}$ is *crossing* if $\mathcal{T}_{\mathcal{O}} \neq \emptyset$ (i.e. it is non-empty), and trades should successively be chosen and execute until the new market $\mathcal{O}'$ has $\mathcal{T}(\mathcal{O}') = \emptyset$. A real market considers a filtered view of the candidate trades, $F(\mathcal{T}_{\mathcal{O}}) \subseteq \mathcal{T}_{\mathcal{O}}$.

Priority is determined by preferring, in order: a strict subset of orders involved[2], followed by minimal cost to the aggressor, followed by (time) priority lexicographically of the orders involved, followed by preferring maximum volume. More formally:

Definition 3: For candidate trades $t, t' \in \mathcal{T}_{\mathcal{O}}$ we say *t has priority over t'* if and only if either:

1. $\sup(t) \subset \sup(t')$ (see footnote 2); or, if they are either equal or incomparable;
2. $\left(\sum_{o \in \mathcal{O}} t(o) \cdot \mathrm{cost}(o)\right) < \left(\sum_{o \in \mathcal{O}} t'(o) \cdot \mathrm{cost}(o)\right)$; or, if they are equal cost;
3. $(\max_{o \in \sup(t) \setminus \sup(t')} \mathrm{prio}(o)) > (\max_{o \in \sup(t') \setminus \sup(t)} \mathrm{prio}(o))$; or, if $\sup(t) = \sup(t')$;
4. $t(o) \geq t'(o)$ for all $o \in \mathcal{O}$.

This total ordering (proof elided) establishes a top priority trade in $\mathcal{T}_{\mathcal{O}}$ (and in fact any finite set of candidate trades).

This appears quite simplistic but most real markets can be represented by the right choice of candidate trade filter F. To give a few examples:

1. A market without implications/baits corresponds to F filtering trades containing more than two orders.
2. The single-implication market which can *always* trade out bait-hitting-bait cases corresponds to F filtering all trades which include more than two distinct combinations.
3. Some markets permit multi-stage implications but do not permit trades being implied outside the spread of a book, here F would inspect other orders to filter such trades.

Finally, a non-crossing market $\mathcal{O}$ where an order x causes $\mathcal{O} \cup \{x\}$ to be crossing, with the top priority trade t having cost zero and $t(x) = \mathrm{qty}(x)$, then the *inverse of x is an implied bait* in $\mathcal{O}$. This procedure is iterated to generate all baits at depth.

2 Efficiently Implementing Perfect-Derived Markets

A key property of $\mathcal{T}_{\mathcal{O}}$ is that it and the baits implied are trivial to compute by simple enumeration. This minuscule implementation is important as a starting point for validation, but the computation, and $\mathcal{T}_{\mathcal{O}}$ itself, is exponential in size.

The BeTefex perfect matcher takes this as the initial verifiable core, compiling the computation of $\mathcal{T}_{\mathcal{O}}$ into a large set of simple logical propositions. From there:

[1]This is left informal, but in the usual case multiply the legs of each o by $t(o)$ and sum up across $\mathcal{O}$ for each contract. There may be additional business requirements, but those are better handled by a filter F, discussed later.

[2] This is largely a trade reporting technicality, where if t is the merge of two trades, $\sup(t) = \sup(t') \cup \sup(t'')$, one of the trades with a subset of the orders involved will be preferred (note that one of t' and t'' will be at least as good as t in cost and time priority). It is not obvious that this behavior is always desirable from a business perspective, and it can be reversed to instead coalesce trades.

- Current research in incremental and parallel propositional logic solvers is leveraged and extended in novel ways to efficiently precompute a compact representation of $\mathcal{T}_{\mathcal{O}}$ in a form where the effect of adding or removing an order can be done almost instantly using the preexisting data.
- The logic machinery is extended by an incrementally built weighted hypergraph orderbook structure as an underlying theory (i.e. entering the realm of SMT solvers, propositional solvers equipped with a theory). As the relevant connections at each price level are extremely sparse this is a huge optimization while safely preserving semantics.
- Infrastructure to compile a new filter F into the logic representation is provided (i.e. compiling a logic representation which computes $F(\mathcal{T}_{\mathcal{O}})$ directly without computing $\mathcal{T}_{\mathcal{O}}$), by optimizing this representation a measure of how much F simplifies solving is given.
- Several families of filters F are provided: optimization filters which shrink $F(\mathcal{T}_{\mathcal{O}})$ without losing volume, or in more aggressive cases without losing top volume (so volume at price levels below the top may be lost), and business-oriented filters which simulates common market behaviors as outlined in Section 1.
- While e.g. equilibrium pricing adds another layer of complexity this general technique (especially incremental computation) can also define auctions with complex implications.
- Finally, sound theoretical underpinnings make it possible to make usually-perfect markets, where full volume is shown best-effort, but implications are disregarded dynamically and intelligently[3] when non-functional requirements dictate. Reenabling implications is a particular difficulty, as the market may already be crossing, requiring complex business decisions better defined in a general solver than encoded in specialized code.

The combination of these pieces produces matchers with well-defined semantics, which can be configured in a flexible way. The compiled logical representation permits not only experimenting with filters, combining various semantics, but also to objectively judge semantics and market complexity by statistically sampling randomized solving (i.e. applying WalkSAT).

Further, by optimizing on the level of a solver high performance, including incremental and predictive precomputation, is achieved without impacting, obscuring, or making unchangeable in the future, the semantics of the market.

[3]Notably implications for the most active orderbooks may not be the right ones to disable, as the difficulty is often generated by low-volume complex combinations. Disabling should happen in a way optimizing volume achieved per solver step taken.