Knapsack problem

Yana Kozak,
Approximation Algorithms Course at Università degli Studi di Perugia,
a.y. 2019/2020

Description

We want to choose which items to put in our backpack, so that we profit the most by taking them with us.

Background

The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name "knapsack problem" dates back to the early works of mathematician Tobias Dantzig - the father of George Dantzig (one of the founders of linear programming - he invented the simplex algorithm)

Complexity

DECISION PROBLEM:

- WEAKLY NP-COMPLETE
  - when inputs are positional numbers (KP being pseudopolynomial)
- STRONGLY NP-COMPLETE
  - unary version of the KP remains NP-complete

OPTIMIZATION PROBLEM:

- NP-HARD

Definition of pseudopolynomial algorithm

An algorithm is said to be pseudopolynomial if its running time is polynomial in n and in one (or several) of the input values. So even a simple problem with a small number of items may have quite large coefficients and hence a very long running time.

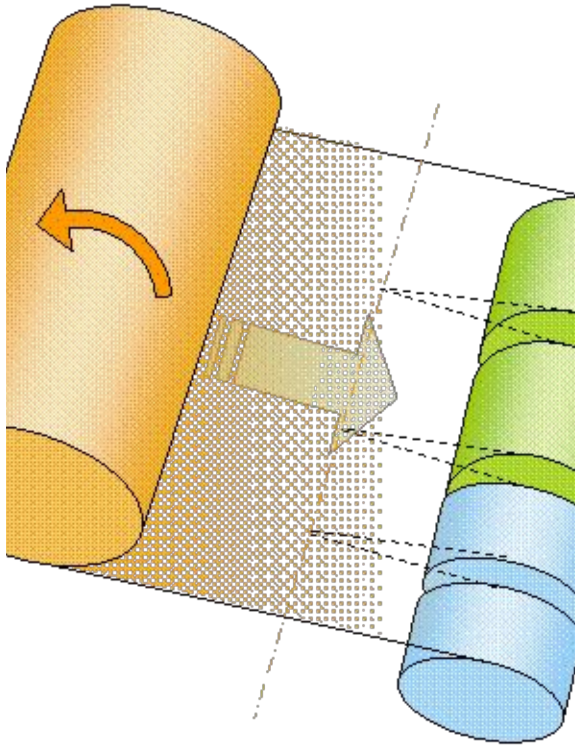
Real-life applications

Knapsack Cryptosystems

Their security is based on the hardness of solving the knapsack problem - hence one theme in research literature is to identify what the "hard" instances of the knapsack problem look like. Even if they remain quite unpopular because simple versions of these algorithms have been broken for several decades, like Merkle-Hellman (1978) - for example -, some are considered secure so far - Nasako-Murakami (2006) is one of those - and they are a good candidate for post-quantum cryptography (not subject to classical cryptoanalysis).

Raw material industry

Finding the least wasteful way to cut standard-sized pieces of raw material into pieces of specified sizes - such as paper rolls or sheet metal - is defined as the cutting-stock problem, which is reducible to the knapsack problem.



Resource allocation

The decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively. Some examples include:

- selection of investments and portfolio
- combinatorial auctions
    - find an allocation of “bundles” of items to bidders
- selection of assets for asset-backed securitization
    - mortgage-based securizations for example
    - selection of assets to be transformed into securities

Construction and scoring of tests

One early application of knapsack algorithms (1973) was in the construction and scoring of tests in which the test-takers have a choice as to which questions they answer and then a knapsack algorithm would determine which subset of problems gives each student the highest possible score.

Formal definition and algorithms

Formal definition

maximize $\sum^{n}_{j=1} v_j x_j$

subject to $\sum^{n}_{j=1} t_j x_j \leq B,$

$x_j \in \{0,1\}, \qquad \forall j \in$

Exact algorithms

DYNAMIC PROGRAMMING:

- PSEUDOPOLYNOMIAL

BRANCH-AND-BOUND TECHNIQUE:

- NO BOUNDS KNOWN A PRIORI

HYBRID APPROACH:

- PSEUDOPOLYNOMIAL

Approximation algorithms

PTAS:

- RUNNING TIME POLYNOMIAL IN N

FPTAS:

- RUNNING TIME POLYNOMIAL IN BOTH N AND 1/ε

Definition of PTAS

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a $(1 + \varepsilon)$-approximation algorithm (for minimization problems) or a $(1-\varepsilon)$-approximation algorithm (for maximization problems).

Definition of FPTAS

A fully polynomial-time approximation scheme (FPAS, FPTAS) is an approximation scheme such that the running time of A_ε is bounded by a polynomial in $1/\varepsilon$.

Approximation algorithms

PTAS:

- GUESSING ITEMS INCLUDED IN THE OPTIMAL SOLUTION
    - and then filling the remaining capacity in some greedy way

FPTAS:

- SCALING THE PROFIT SPACE
    - applying dynamic programming

Dynamic algorithm

Variables and properties

- $A(j)$ for $j = 1, \ldots, n$ is a list of pairs (t,w)
- $S \subseteq \{1, \ldots, j\}$ and $\sum_{i \in S} s_i = t \leq B$ and $\sum_{i \in S} v_i = w$
- (t, w) dominates (t', w') if $t \leq t'$ and $w \geq w'$
- in any list, no pair dominates another one:
    - $A(j)$ is of the form $(t_1, w_1), \ldots, (t_k, w_k)$ with $t_1 < t_2 < \cdots < t_k$ and $w_1 < w_2 < \cdots < w_k$

Pseudocode

*making use of optimal substructure

```
$A(1) \leftarrow \{(0,0),(s_1,w_1)\}$
**for** $j \leftarrow 2$ to $n$ **do**
    $A(j) \leftarrow A(j-1)$
    **for** each $(t,w) \in A(j-1)$ **do**
        **if** $t + s_j \leq B$ **then**
            Add $(t+s_j, w+v_j)$ to $A(j)$
    Remove dominated pairs from $A(j)$
**return** $\max_{(t,w)\in A(n)} w$
```

Pseudopolynomial running time

Dynamic algorithm takes $O(n \times \min(B, V))$ time. This is not a polynomial-time algorithm, since we assume that all input numbers are encoded in binary. Thus the size of the input number B is essentially $\log_2 B$.

Approximated FPTAS algorithm

FPTAS algorithm

$$\boxed{\begin{array}{l} M \leftarrow \max_{i \in I} v_i \\ \mu \leftarrow \epsilon M / n \\ v_i' \leftarrow \lfloor v_i / \mu \rfloor \text{ for all } i \in I \end{array}} + \boxed{\text{Dynamic algorithm}} = \boxed{\text{FPTAS approximation algorithm for knapsack problem}}$$

Prooving that running time is bounded

The introduced error

We measure value in multiples of **μ**

each value is inaccurate by at most **μ**

a solution has its value changed by at most **nμ**

Bounding error

we want our error to be at most **ε** times the optimal value

a lower bound on the optimal value is the maximum value item **M**

$n\mu = \varepsilon M$, or in other words, to set **$\mu = \varepsilon M/n$**

Running time

$$V' = \sum_{i=1}^{n} v_i' = \sum_{i=1}^{n} \left\lfloor \frac{v_i}{\epsilon M/n} \right\rfloor = O(n^2/\epsilon)$$

$$O(n \min(B, V')) = O(n^3/\epsilon)$$

Prooving the approximation

Inequalities and definitions

- S is the set of items returned by the algorithm
- O is an optimal set of items

$$\mu v_i' \leq v_i \leq \mu(v_i' + 1)$$

$$\mu v_i' \geq v_i - \mu$$

Proof

$$\begin{aligned}
\sum_{i\in S} v_i &\geq \mu \sum_{i\in S} v_i' \\
&\geq \mu \sum_{i\in O} v_i' \\
&\geq \sum_{i\in O} v_i - |O|\mu \\
&\geq \sum_{i\in O} v_i - n\mu \\
&= \sum_{i\in O} v_i - \epsilon M \\
&\geq \mathrm{OPT} - \epsilon\,\mathrm{OPT} = (1-\epsilon)\,\mathrm{OPT}
\end{aligned}$$

Variations

About variations...

There is a huge amount of different variations of the knapsack problem and often a specific problem is treated only in one or two papers

Generalizations

- Multi-objective knapsack
  - problems frequently addressed include portfolio and transportation logistics optimizations
  - different types of domination (Pareto)
- Multidimensional knapsack
  - the weight of knapsack item is given by a D-dimensional vector
- Quadratic Knapsack
  - an extra profit that can be earned if two items are selected
- Sum of subsets
  - for each item the profit and weight are equal
- Multiple knapsack
  - we have many knapsacks
- Multiple-choice knapsack
  - the items are subdivided into classes and exactly one item must be taken from each class
- Set-union knapsack
  - the profit is assigned to sets, while weight is assigned to the elements inside those sets

Knapsack-like problems

- Bin packing
  - we have containers of the same size, and we wish to pack all items in as few containers as possible
- Cutting-stock problem
  - we have "patterns" that need a predefined quantity of specific items
- Strongly correlated KP
  - a generalization of sum-of-subsets
- Nested knapsack
  - can occur as subproblems of an important type of the cutting stock problem where there are successive cutting stages
- Collapsing knapsack
  - the knapsack capacity is a non-increasing function of the number of items included.
  - it has applications in satellite communication where transmissions on the band require gaps between the portions of the band assigned to each user

Knapsack-like problems

- Change making problem
  - make change using as few coins as possible
- Expanding knapsack
  - as we “buy” our items, we receive “discounts”, so we can buy still other items, without exceeding our original budget
- Precedence constraint knapsack
  - we can put the items in our knapsack only in a specified order
- Non-linear knapsack
  - the costs resp. weights are replaced by linear functions
- Inverse parametric knapsack
  - the costs resp. weights are repaced by linear functions depending on a parameter ***t***
  - find the smallest parameter such that the optimal solution value of the knapsack problem is equal to a prespecified solution value

Knapsack-like problems

- Maximum density knapsack
  - density is expressed as profit over weight ratio
- Maximum weight bipartite matching
  - in a bipartite graph vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V
  - matching or independent edge set in a graph is a set of edges without common vertices
- Min knapsack
  - instead of maximization objective function we have a minimization obj. func.
- Max-Min knapsack
  - the profits and the weights may assume values in a predetermined range: find a solution which minimizes the total possible weight and maximizes the total possible profit
- Equality knapsack
  - the total weight of the items we put into our knapsack must be equal to the knapsack's capacity

Recent variants: dealing with uncertainty

- **Robust knapsack**
  - some of the input parameters may attain arbitrary values in a given interval
  - a greedy heuristic is available
- **Interdiction knapsack**
  - interdiction game is a special case of non-cooperative bilevel games: we have a two-person, zero-sum sequential game and the players have two opposite objective functions
  - an iterative MIP algorithm is available
  - corporate strategy problem
- **Online knapsack**
  - the items are unknown at the beginning and arrive one after the other: it has to be decided immediately whether to pack the arrived item
  - deterministic/stochastic/time dependent (dynamic)/removable/advice model/semi-online models
- **Stochastic knapsack**
  - the input data are modelled as stochastic variables
  - tests in uniformly random data suggest achievability of polynomial expected running time

Open problems

Open problems - examples

- Improving time/space complexity of the known algorithms
- Find suitable restrictions on the pure online formulation which make sense from a real-world point of view

Algorithmical techniques

Algorithmical techniques

- Finding the split item in linear time
  - the first item that we fail to pack
  - it is trivial to find s in O(n) time if the items are already sorted in decreasing order of their efficiency
  - for very large item sets it is interesting to find s in linear time without the O(n log n) effort for sorting the items.
- Reduce the size of a KP before it is solved
  - Variable reduction considerably reduces the observed running time of algorithms for nearly all instances occurring in practice
  - by the way it does not decrease the theoretical worstcase running time as instances may be constructed where the reduction techniques have no effect.

Algorithmical techniques

- Storage Reduction in Dynamic Programming
  - The method can be useful in general for problems where it takes more effort to compute the actual optimal solution set than the optimal solution value.
  - a recursive divide and conquer strategy is used for this purpose
- Dynamic Programming with Lists
  - the following technique does not change the worst-case complexity, but it may considerably improve the computation time for practical instances
  - the list is made of states where each state is a pair of two integers, where W denotes a given capacity, and fl denotes the maximum profit sum obtainable for this capacity when considering the subproblem defined on the first j items
  - a list may be pruned by using the concept of dominance

Algorithmical techniques

- Combining Dynamic Programming and Upper Bounds
  - derive upper bounds on every solution value of a subproblem of the dynamic programming algorithm
  - the algorithm will still have the worst-case complexity of the dynamic programming algorithm while in the best-case it may exploit upper bounds and prune the search considerably as in branch-and-bound.
  - both the previous approach and this one are based on an enumeration of the variables, where dynamic programming is using dominance to improve the complexity, while branch-and-bound is using bounding to prune the search.

Algorithmical techniques

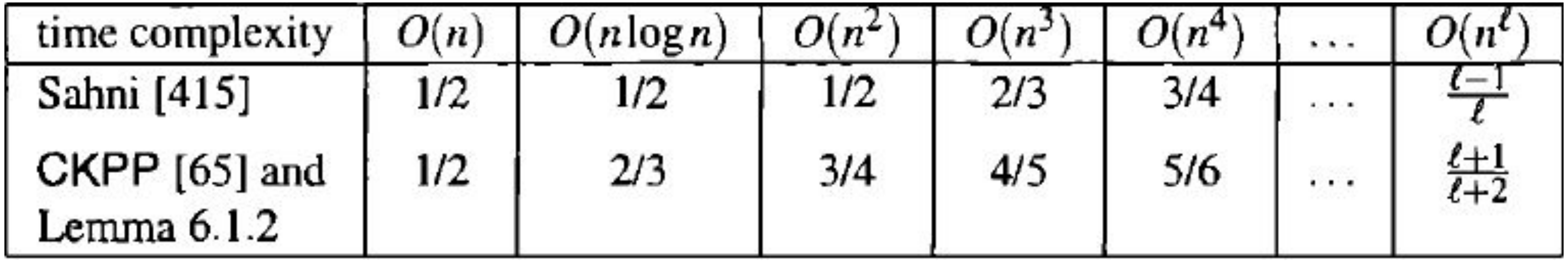
- Balancing
  - an optimal solution is balanced, and hence the dynamic programming recursion may be limited to consider only balanced states
  - a balanced solution is a solution which is sufficiently filled, i.e. the weightsum does not differ from the capacity by more than the weight of a single item.
  - for knapsack problems balancing yields a running time complexity of $O(n\, p_{max} w_{max})$ which may be attractive if the profits and weights of the items are not too large
  - The split solution s is a balanced filling - operations that we can perform on it, keeping the solution balanced, are:
    - Balanced insert:
      - if b>s and the capacity has not been exceeded yet
    - Balanced remove:
      - if a<s and the capacity has been exceeded

Algorithmical techniques

- Word RAM Algorithms
  - the use of parallelism at bit level for solving NP-hard problems through dynamic programming
- Relaxations
  - a technique for deriving an upper bound
- Lagrangian Decomposition
  - splitting the problem into a number of independent problems which can be solved (more) efficiently
  - approximates a difficult problem of constrained optimization by a simpler problem
  - inequality constraints are relaxed, but a penalization is introduced for violations of inequality constraints
  - some “original” inequality constraints could be moved inside the objective function of the new, simpler problem formulation - and represent the penalization cost
- The Knapsack Polytope
  - items are represented as n-dimensional vectors
  - a polyhedron is a set of vectors which satisfy finitely many linear inequalities (defining our specific KP instance) - a bounded polyhedron is called polytope.
  - for a given polyhedron P it is interesting to know which of the satisfied inequalities are really necessary for describing P - and thus to reduce our inequality system

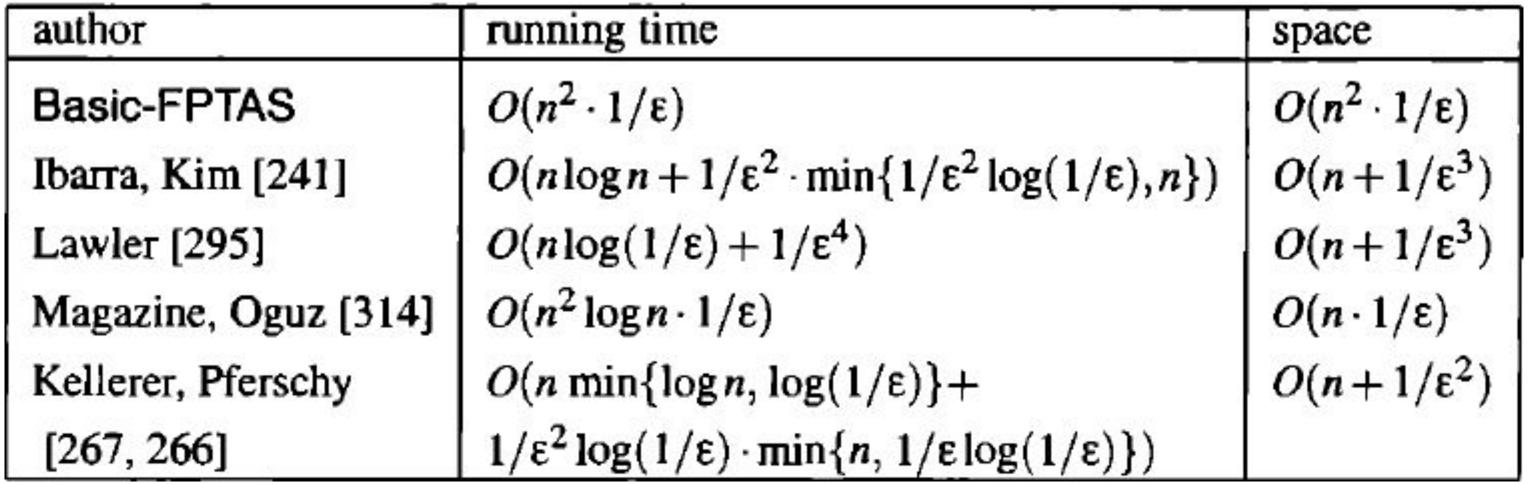
Performance comparison

*tables taken from “Knapsack Problems”, by Kellerer and Pfershy [2]



time complexity	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(n^4)$	...	$O(n^\ell)$
Sahni [415]	1/2	1/2	1/2	2/3	3/4	...	$\frac{\ell-1}{\ell}$
CKPP [65] and Lemma 6.1.2	1/2	2/3	3/4	4/5	5/6	...	$\frac{\ell+1}{\ell+2}$

**guaranteed approximation ratios of PTAS for (KP)



author	running time	space
Basic-FPTAS	$O(n^2 \cdot 1/\varepsilon)$	$O(n^2 \cdot 1/\varepsilon)$
Ibarra, Kim [241]	$O(n \log n + 1/\varepsilon^2 \cdot \min\{1/\varepsilon^2 \log(1/\varepsilon), n\})$	$O(n + 1/\varepsilon^3)$
Lawler [295]	$O(n \log(1/\varepsilon) + 1/\varepsilon^4)$	$O(n + 1/\varepsilon^3)$
Magazine, Oguz [314]	$O(n^2 \log n \cdot 1/\varepsilon)$	$O(n \cdot 1/\varepsilon)$
Kellerer, Pferschy [267, 266]	$O(n \min\{\log n, \log(1/\varepsilon)\} + 1/\varepsilon^2 \log(1/\varepsilon) \cdot \min\{n, 1/\varepsilon \log(1/\varepsilon)\})$	$O(n + 1/\varepsilon^2)$

**Complexities of fully polynomial approximation schemes for (KP)

Some lower bounds...

For KP a lower bound of $\Omega(n^2)$ holds for the number of arithmetical operations necessary to decide KP [3].

Restrictions on the online knapsack

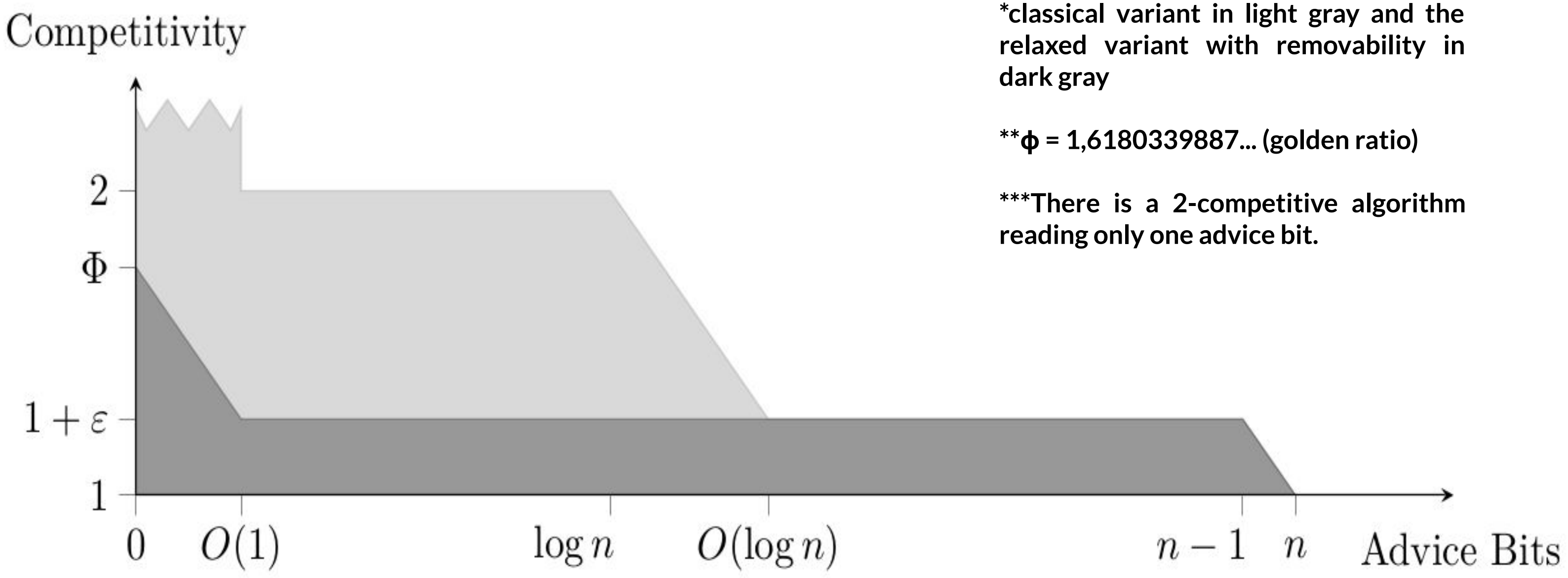
Advice model & online algorithms: What amount of information really lacks?

Very small advice already suffices to significantly improve over the best deterministic algorithm. Moreover, to achieve the same competitive ratio as any randomized online algorithm, a logarithmic number of advice bits is sufficient. On the other hand, to obtain optimality, much larger advice is necessary [4].

Advice model and knapsack problem

- a single random bit allows for a competitive ratio of 2, and any further amount of randomness does not improve this.
- in a resource augmentation model, i. e., when allowing a little overpacking of the knapsack, a constant number of advice bits suffices to achieve a near-optimal competitive ratio.
- in the weighted version of the problem proving that, with O(log n) bits of advice,we can get arbitrarily close to an optimal solution and, using asymptotically fewer bits, wea re not competitive [5].

Classical and Removable online KP - advice complexity comparison



*classical variant in light gray and the relaxed variant with removability in dark gray

**ɸ = 1,6180339887... (golden ratio)

***There is a 2-competitive algorithm reading only one advice bit.

Classical online KP and Advice model

- one of the prime examples of a problem with an interesting advice complexity. First, just a single advice bit brings with it a jump from non-competitivity to a 2-competitive algorithm.
- Once the threshold of log(n − 1) is surpassed, logarithmic advice allows for a competitive ratio that is arbitrarily close to 1.
- The jump to optimality, finally, requires at least n − 1 advice bits.

Removable online KP and Advice model

The competitivity starts from the golden ratio when no advice is given. It then drops down in small increments to (1 + ε) for a constant amount of advice already, which requires logarithmic advice in the classical version. Optimality still requires one full advice bit for every single item in the instance as before [1].

Removable online KP and Advice model: curiousities

These results are particularly noteworthy from a structural viewpoint for the exceptionally slow transition from near-optimality to optimality; such a steep jump up from constant to full linear advice for just an infinitesimally small improvement is unique among the online problems examined so far

Advice model & online algorithms: general open problems

In general, the advice complexity is somehow orthogonal to randomization. It remains as an open problem to find more connections between these complexity measures, e. g., to somehow characterize classes of online problems where small advice is sufficient to keep up with randomized algorithms. Furthermore, it might be interesting to consider randomized online algorithms with advice.

Possible applications

Possible application: Travel agency

Travel agency that offers a holiday plan to its client, who disposes of a specific budget and has predetermined preferences about holiday activities.

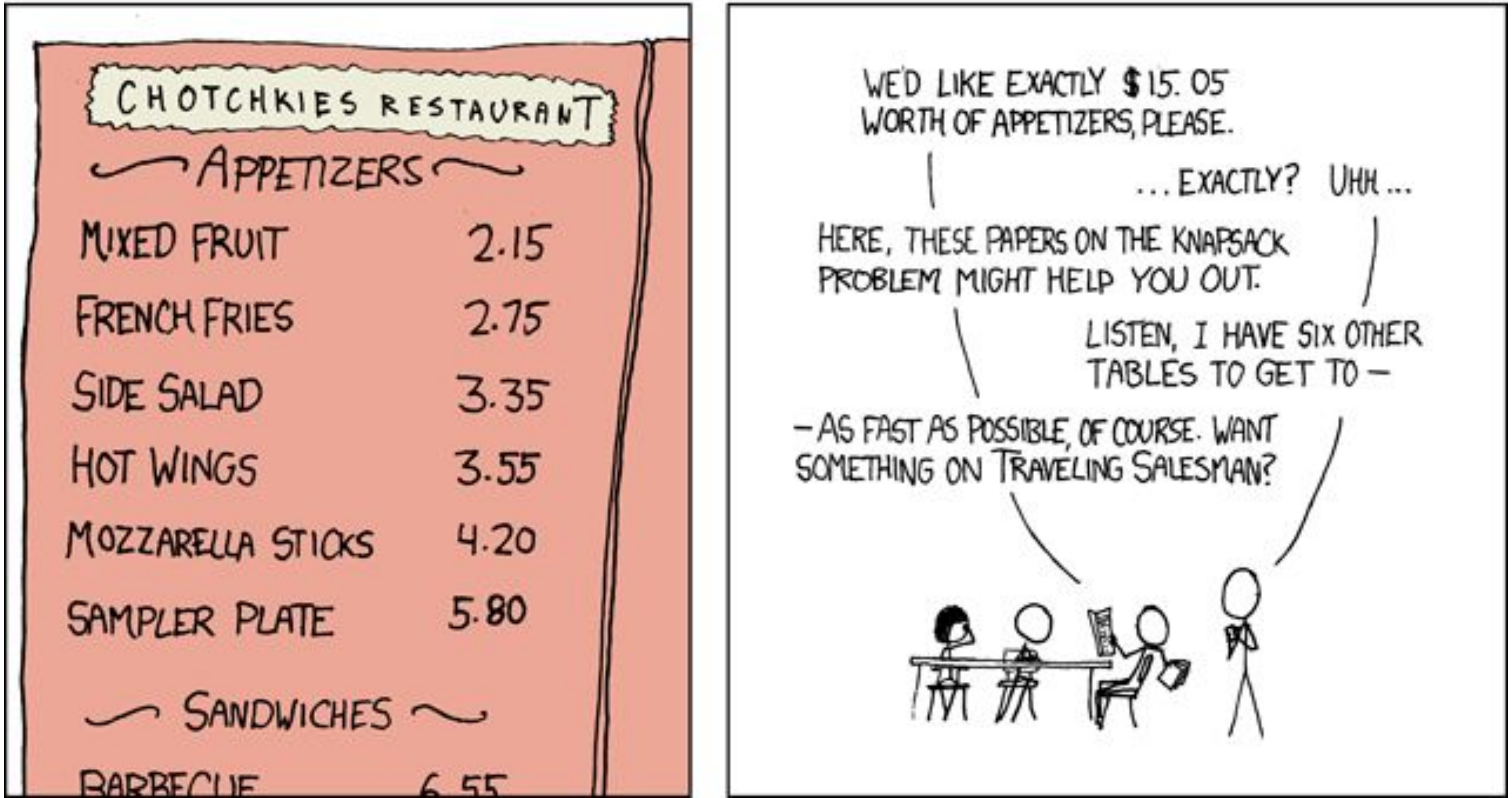
Possible application: Personal organizer

Quite a futuristic idea, requiring significant breakthroughs in Natural Language Processing and in in Artificial Intelligence. A person saves into her/his organizer the thing she/he has to see to and the organizer, basing itself on some databanks and on the preferences and behaviour of its owner, creates a schedule - trying to fill free-time windows of its owner as well as possible.

References

1. Böckenhauer, H. J., Dreier, J., Frei, F., & Rossmanith, P. (2020). Advice for Online Knapsack With Removable Items. *arXiv preprint arXiv:2005.01867*.
2. Kellerer, Hans, Ulrich Pferschy, and David Pisinger. "Knapsack problems. 2004."
3. L. Blum, F. Cucker, M. Shub, and S. Smale. Complexity and Real Computation. Springer. 1997.
4. Böckenhauer, Hans-Joachim & Komm, Dennis & Královič, Rastislav & Královič, Richard & Mömke, Tobias. (2009). On the Advice Complexity of Online Problems.
5. Böckenhauer, Hans-Joachim, et al. "On the advice complexity of the knapsack problem." 2012.

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS


CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Thank you for your attention