Polynomial-Time Combinatorial Bandits

Computationally Tractable Reinforcement Learning in Complex Environments

Thibaut Cuvelier

Combinatorial bandits

- **Machine learning**: how do computers learn from data?
  - **Supervised learning**: static relation between given input and output
    - *Sample task:* predict the traffic on A6 highway
    - *Data*: previous traffic measurements
  - **Reinforcement learning**: act in an environment
    - *Sample task*: play a video game
    - *Data*: actions taken in the game, final score
  - ► Take **sequential decisions** based on experience
  - **Combinatorial bandits**: special case of reinforcement learning where decisions have a structure
    - *Sample task*: choose a route from home to work
    - *Data*: time taken the previous days, corresponding paths

Combinatorial bandit: navigator systems

- How do systems like Waze determine the best paths?
  - Easy to do once the congestion is known everywhere!
- **The catch**: how to know the congestion?
  - Send drivers on the roads to estimate the congestion!
- Exploration-exploitation dilemma:
  - A driver that “explores” a poor path may be unsatisfied
  - A driver that benefits from the others’ exploration is satisfied
- Formulate this problem as a combinatorial bandit
  - An action is a path from the user’s position to their destination
  - The combinatorial set is the set of paths in the graph of roads

Combinatorial bandits: some vocabulary

- A bandit algorithm plays **actions** in a combinatorial set $\mathcal{X}$
  - *Path in a graph:* Waze, computer-network routing
  - *Matching:* display ads on a Webpage
- Playing an action yields some (random) **reward**
  - *Matching:* 1 if user clicks, 0 otherwise
  - *Path in a graph:* inverse of time to traverse a road
- Actions are composed of *d* **subarms**
  - *Path in a graph:* edges of the graph (e.g., roads, network link)
  - *Matching:* association between two nodes (e.g., ad position and content)
- For each bandit problem, there is an **optimum action**
  - The difference in reward between *one* played action and the optimum action is the **gap** (symbol: Δ)
  - The total difference in reward between *all your actions* and playing the optimum action all the time is the **regret**

Why are combinatorial bandits hard?

- Combinatorial problems are hard

  Goal: find the best solution for **known costs**

  - Many interesting exceptions, though:
    - Shortest path: network routing, GPS navigators
    - Matchings: matchmaking

- Uncertain combinatorial problems are extremely hard

  Goal: find the best solution for **unknown costs**

  - Major tool: nonlinear combinatorial optimisation
  - Close to no exceptions… in general

- Up to now, general belief that there is a trade-off between time complexity and regret performance

Outline

- AESCB: a first state-of-the-art algorithm
- GLPG: reaching the lower bound
- The underlying optimisation algorithms

ESCB: combinatorial bandits with confidence sets

- General technique for reinforcement learning:
  play the action with the largest *upper bound on the reward*
  - Combination of the average reward and a confidence bonus
  - "Optimism in the face of uncertainty"
- One example for combinatorial bandits: ESCB

$$\max_{\boldsymbol{x} \in \mathcal{X}} \widehat{\boldsymbol{\theta}}^T \boldsymbol{x} + \sqrt{\boxed{\boldsymbol{t}}^T \boldsymbol{x}}$$

Low for actions played a lot
High for actions barely played

Find a combinatorial solution maximising…

… the average reward…

… and the confidence bonus

Close to the true value if played a lot
Potentially far off if barely played

AESCB: an efficient implementation

- Even for easy combinatorial problems, ESCB cannot be implemented efficiently ("in polynomial time")
- **Idea**: approximate the problem with budgeted optimisation
  - Budget *s*: value for the nonlinear term
  - Effect: linearises the objective as a constraint

$$\begin{array}{rl} \max & \hat{\boldsymbol{\theta}}^T \boldsymbol{x} \\ \text{subject to} & \boldsymbol{t}^T \boldsymbol{x} \geq s \\ & \boldsymbol{x} \in \mathcal{X} \end{array}$$

Two problems:

- Which values for the budget?
- How to solve budgeted problems?

AESCB: an efficient implementation

Two problems:

- Which values for the budget?
  - Only allow integers as coefficients
  - ►Use scaling and rounding!

- How to solve budgeted problems?
  - Write a dedicated algorithm for each combinatorial problem
  - This technique works well for many problems:
    knapsacks, shortest paths, spanning trees, matchings, etc.
  - The dedicated algorithm is sometimes exact, at least approximate
  - The approximation factor has a constant impact on the regret

AESCB in practice

- AESCB is successful *in practice* if:
  - It runs faster than ESCB
  - It runs *much faster* than ESCB *in large dimensions*
  - Its regret is close to that of ESCB (slightly worse due to approximation)
- Compare it to an advanced implementation of ESCB
  - Use the nonlinear features of CPLEX (MISOCP), a state-of-the-art optimisation solver
  - Standard formulation of the combinatorial sets
- Also compare to other algorithms: CUCB and Thompson sampling (TS)
  - Faster, but poorer performance guarantees than (A)ESCB

AESCB in practice: runtime

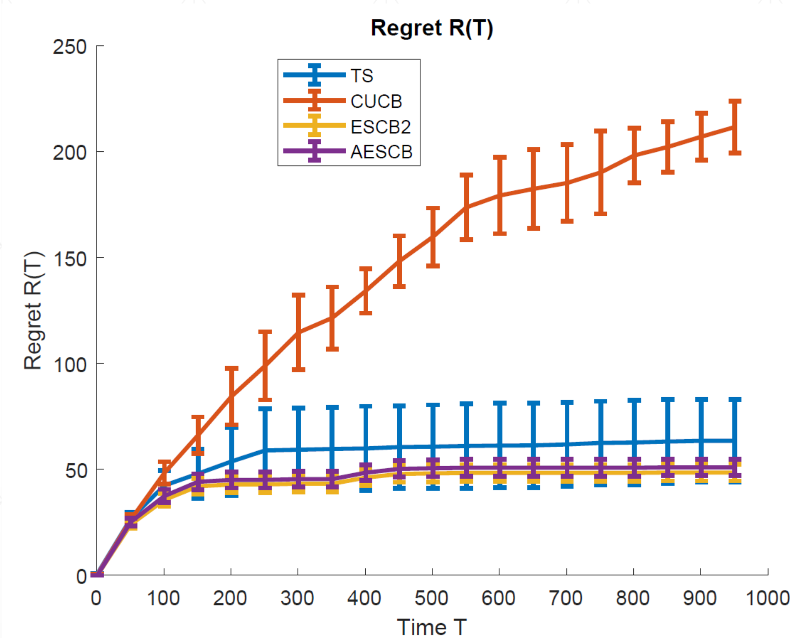
- In low dimension, both ESCB and AESCB run too fast
- Thus, in higher dimension:

Combinatorial set	ESCB	AESCB
At most 16 elements among 50	1.24 ± 0.03 s	**0.10 ± 0.03 s**
Path in a 190-node graph	0.11 ± 0.04 s	**0.05 ± 0.00 s**
Spanning tree in a 190-node graph	0.20 ± 0.03 s	**0.04 ± 0.01 s**
Matching in a 25-25-bipartite graph	0.26 ± 0.06 s	**0.18 ± 0.01 s**

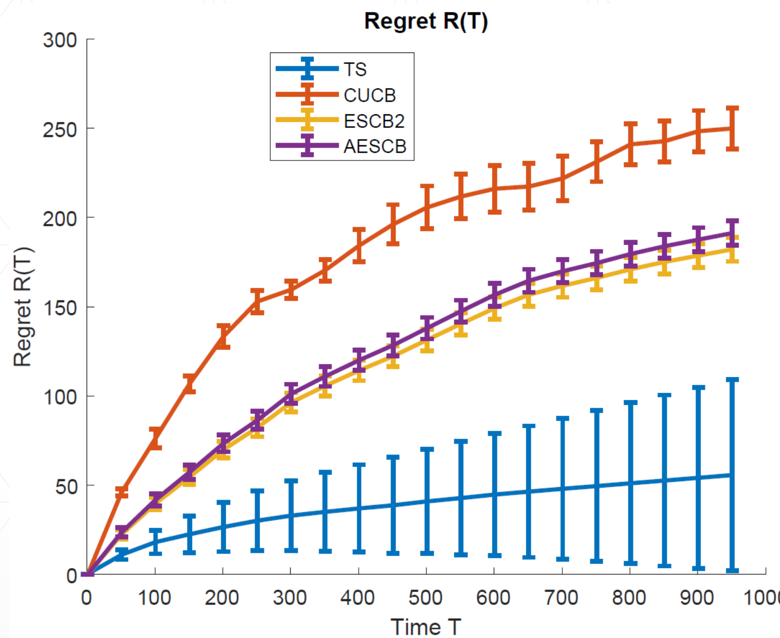
- AESCB is always faster!

AESCB in practice: regret

- And in terms of regret?



Matching



Spanning tree

- Thompson sampling has a very high variance
- CUCB is the worst algorithm
- AESCB is extremely close to ESCB

GLPG: asymptotically optimal combinatorial bandits

- A technique that is very specific to bandit problems
  - In the long term, what is the minimum degree of exploration needed to ensure that only the best solutions are played?
  - Based on a mathematical property of the problem called the *Graves-Lai bound*

Frequency to play ***x***

Gap of ***x***

$$\min \quad \sum_{\boldsymbol{x} \in \mathcal{X}} \alpha_{\boldsymbol{x}} \Delta_{\boldsymbol{x}}$$

Total regret

$$\text{subject to} \quad \sum_{i=1}^{d} \frac{x_i}{\sum_{\boldsymbol{y} \in \mathcal{X}} y_i \, \alpha_{\boldsymbol{y}}} \leq \Delta_{\boldsymbol{x}}^2 \qquad \forall \boldsymbol{x} \in \mathcal{X}$$

Distinguish optimum solutions from others

$$\alpha_{\boldsymbol{x}} \geq 0 \qquad \forall \boldsymbol{x} \in \mathcal{X}$$

The Graves-Lai bound

- Intuitive meaning:
  - If you explore less than this: you might think a solution is optimal when it is not
  - If you explore more than this: too much regret for the same level of confidence you have found the optimum solution

- Computational problems:
  - Large number of variables
  - Large number of constraints (but convex)
- **GLPG to the rescue!**

The crux of GLPG

- The Graves-Lai problem has a lower *intrinsic* dimensionality
  - *Change variables:* use subarm frequency as variables
  - *Use a nonsmooth constraint:* instead of many smooth constraints
    More precisely: replace ∀ by *max*
- The new formulation:

$$\begin{aligned} \min \quad & \boldsymbol{q}^T \boldsymbol{w} \\ \text{subject to} \quad & \max_{\boldsymbol{x}\in\mathcal{X}} \left\{ \sum_{i=1}^{d} \frac{x_i}{w_i} - \Delta_{\boldsymbol{x}}^2 \right\} \leq 0 \\ & \boldsymbol{M}\,\boldsymbol{w} = \boldsymbol{0} \end{aligned}$$

GLPG: *projected* subgradient

Final algorithm:

- Penalise the nonsmooth constraint
  - If the weight λ is large enough, the constraint will be satisfied
  - New problem: convex nonsmooth objective, linear constraints

$$\begin{array}{ll} \min & \boldsymbol{q}^T\,\boldsymbol{w} + \lambda\left[\max_{\boldsymbol{x}\in\mathcal{X}}\left\{\sum_{i=1}^{d}\frac{x_i}{w_i} - \Delta_{\boldsymbol{x}}^2\right\}\right]^+ \\ \text{subject to} & \boldsymbol{M}\,\boldsymbol{w} = \boldsymbol{0} \end{array}$$

- Use a *projected* subgradient method

GLPG: complexity

Three important parts to guarantee a polynomial time complexity:

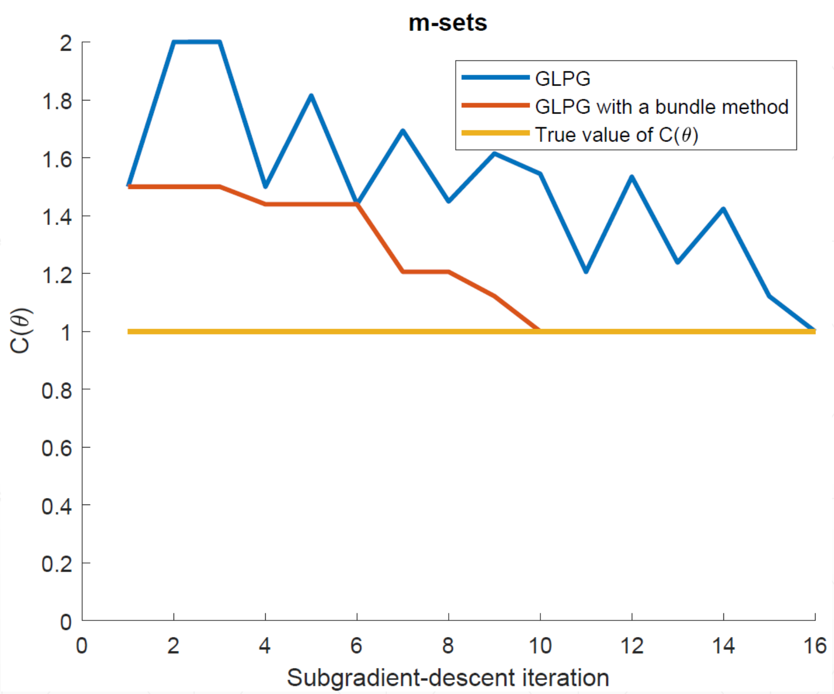
1. Evaluate the objective function and a subgradient
   - Use the same technique as ESCB!
2. Convergence of the subgradient method
   - We slightly generalise known convergence results
   - Approximate budgeted optimisation is not a problem
3. Convergence of the projection operator
   - Minimise a smooth convex objective with linear constraints
   - Known result from the literature (e.g., interior-point method)

GLPG in practice

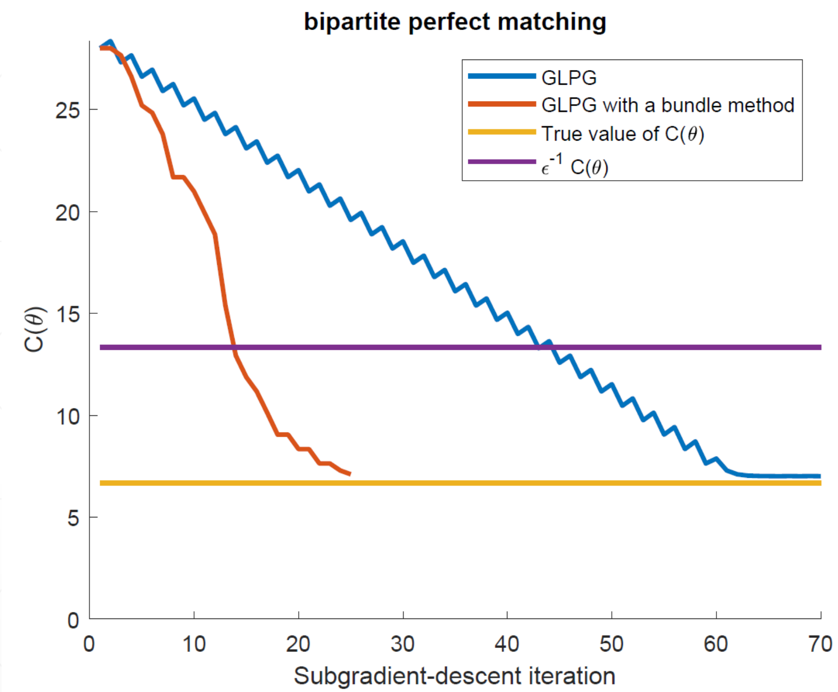
- GLPG is successful *in practice* if:
  - It runs fast (but not as fast as AESCB)
  - Its result is close to the true value of the Graves-Lai bound (considering the approximation ratio, if need be)
- Compare it to an advanced implementation of the Graves-Lai bound
  - Work on the reformulation with fewer variables
  - Use the nonlinear features of CPLEX (SOCP), a state-of-the-art optimisation solver
  - Constraint generation for the many convex constraints
- Compare to GLPG with a bundle method
  - Converges faster than the subgradient method

GLPG in practice: convergence speed

- How fast does GLPG converge?
  - Each subgradient/bundle iteration brings it closer to the optimum



Choose one element among two



Matching in a 2-2-bipartite graph

► Converges in few iterations (especially bundle)

► Approximation in the budgeted subproblem is not an issue

Optimising a class of nonlinear functions

- Both AESCB and GLPG rely on the same subproblem:
  - A new approximation scheme for a class of nonlinear combinatorial optimisation problems
  - Based on the building block of budgeted *linear* optimisation
- Considered objective functions:

$$f(\boldsymbol{x}) = \boldsymbol{a}_0^T \boldsymbol{x} + \sum_i f_i\left(\boldsymbol{a}_i^T \boldsymbol{x}\right)$$

  - Where the f_i are invertible unary functions (i.e. not necessarily convex or concave)
- By itself, our approximation scheme does not always yield polynomial-time algorithms!

Optimising a class of nonlinear functions

- Consider that the f_i are increasing
- Our "decomposition" technique:
  - Find the range of values for the f_i
  - Discretise this range (up to some precision ε):

$$\phi'_i \mapsto \varepsilon \left\lfloor \frac{\phi_i}{\varepsilon} \right\rfloor$$

  - Optimise a series of budgeted problems:

$$\begin{aligned} \max \quad & \boldsymbol{a}_0^T \, \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{a}_i^T \, \boldsymbol{x} \geq f_i^{-1}(\phi'_i), \qquad \forall i \\ & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

iterating over the values of ϕ'_i for each nonlinear term *i*

Optimising a class of nonlinear functions

- When does this scheme yield a polynomial-time algorithm?
  - If the range of values for ϕ_i' is bounded by a polynomial
  - If optimising the budgeted problem can be done in polynomial time
  - ▶ Many interesting cases where both happen!
- For instance: knapsacks
  - Pick any number of items (total weight less than a fixed threshold) to maximise the total value of the chosen items
  - Standard technique: dynamic programming

Maximum value: items 1 to *i*, maximum weight *w*

$$m_{i,w} = \max\{m_{i-1,w}\ ,\quad v_i + m_{i-1,w-w_i}\}$$

Don't take item *i*

Take item *i*

Value of item *i*

Weight of item *i*

Optimising a class of nonlinear functions

- Standard technique for knapsacks:

$$m_{i,w} = \max\{m_{i-1,w}\ ,\quad v_i + m_{i-1,w-w_i}\}$$

- Generalised algorithm for one budget:

Maximum value: items 1 to *i*, maximum weight *w*, minimum budget *s*

$$m_{i,w,s} = \max\{m_{i-1,w,s}\ ,\quad v_i + m_{i-1,w-w_i,s-s_i}\}$$

Don’t take item *i*

Take item *i*

Budget of item *i*

- When the weights and budgets are properly bounded integers, the time complexity is polynomial

Conclusion

- Combinatorial bandits are a hard computational problem
- AESCB is a fast algorithm that achieves very low regret
- GLPG allows to compute the lower bound in polynomial time
  - It can be used to power bandit policies like OSSB
- We solve the computational aspects of combinatorial bandits with a novel methodology for nonlinear optimisation
  - Based on the concept of budgeted optimisation

References

- ESCB:
  Richard Combes, Mohammad Sadegh Talebi Mazraeh Shahi, and Alexandre Proutière. Combinatorial bandits revisited. In *Advances in Neural Information Processing Systems*, pages 2116–2124, 2015
- AESCB:
  Thibaut Cuvelier, Richard Combes, and Éric Gourdin. Statistically Efficient, Polynomial-Time Algorithms for Combinatorial Semi-Bandits. In *ACM SIGMETRICS 2021*, Beijing (China), 2021
- OSSB:
  Richard Combes, Stefan Magureanu, and Alexandre Proutiere. Minimal exploration in structured stochastic bandits. In *Advances in Neural Information Processing Systems*, pages 1763–1771, 2017
- GLPG:
  Thibaut Cuvelier, Richard Combes, and Éric Gourdin. Asymptotically optimal strategies for combinatorial semi-bandits in polynomial time. In *Algorithmic Learning Theory 2021*
- GLPG numerical results:
  Thibaut Cuvelier. Polynomial-Time Algorithms for Combinatorial Semibandits: Computationally Tractable Reinforcement Learning in Complex Environments. CentraleSupélec (université Paris-Saclay)


DO YOU HAVE ANY QUESTIONS FOR ME?