Google

Consistency in Solvers.

Or what is the expected answer of a solver when you run it several times on the same problem.

Bertrand LeCun
Operations Research
bertrandlc@google.com

Table of contents

Google Operations Research Team

- We:
  - model problems,
  - choose the best solver,
  - ...and help develop specific solutions
- We also develop our own solvers
  - ...some of which are open sourced as OR-tools: https://github.com/google/or-tools

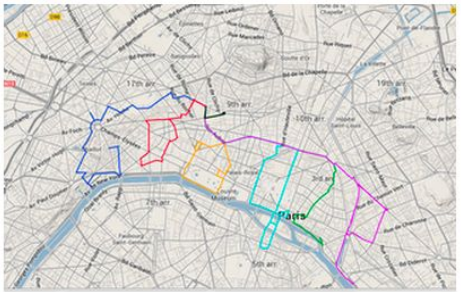
Consulting (and Coding!): project examples

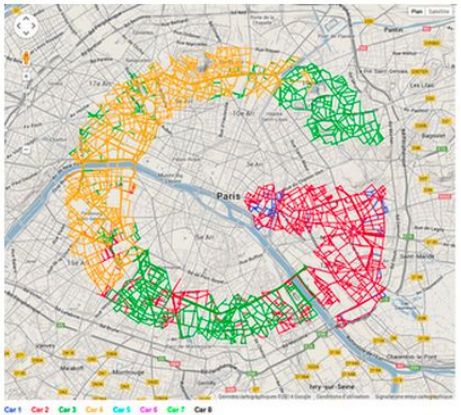
- Network: capacity planning, routing.
- Machines: bin packing, scheduling.
- Replication: placement, synchronization.
- Shortest path in Maps.
- Arc routing.
- Traveling salesman problem in Maps API.
- Ads.
- Machine learning.
- Search.

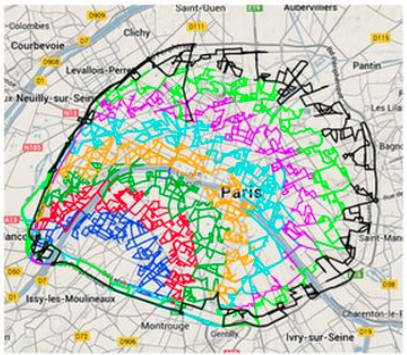
Street View

We’re driving cars down all roads on Earth:

- Taking pictures in a 360° view
- With a fleet of cars
- Decide the number



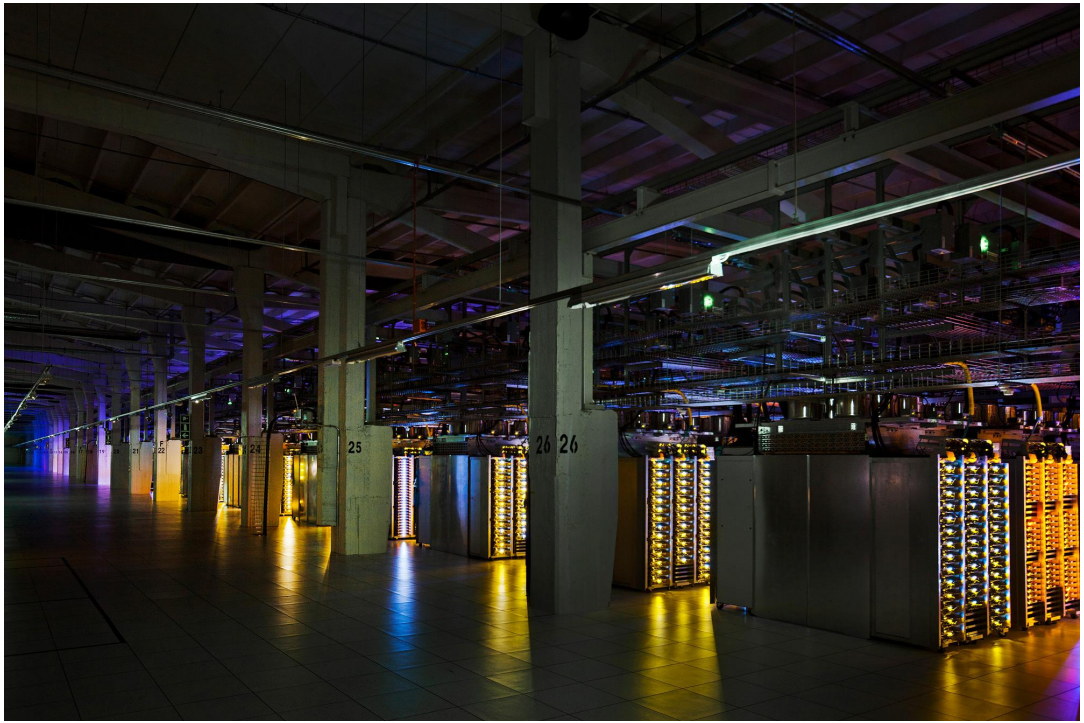




Place Racks in a data center

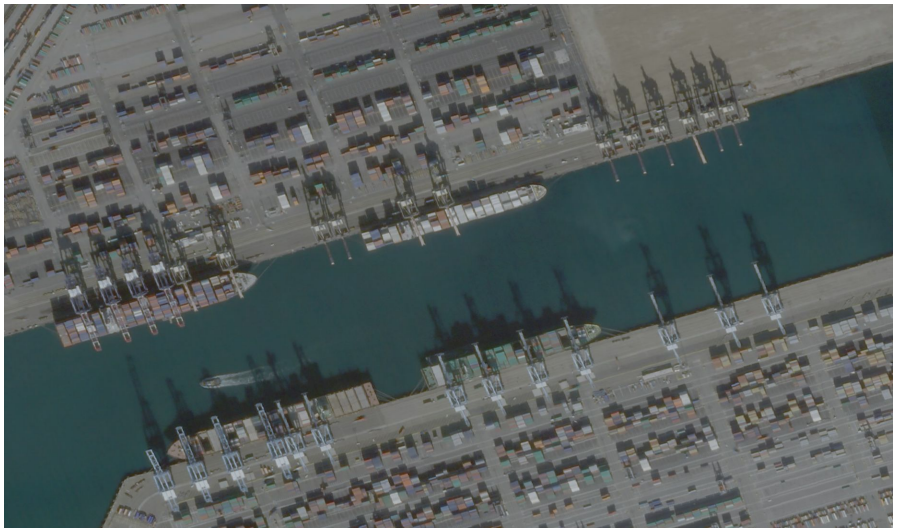
Respect constraints

- On power,
- On cooling
- ...



Skybox

- Set of satellites.
- Each can take pictures.
- Some pictures are requested.
- Schedule satellite operations so that the total value of delivered image collections is as high as possible.



Loon project

Project Loon aims to bring universal Internet access using a fleet of high--altitude balloons equipped with LTE transmitters. Circulating around the world, Loon balloons deliver Internet access in areas that lack conventional means of Internet connectivity.

Develop solvers: more long term.

- Constraint programming solver (Search, LNS, portfolio)
- SAT solver (Search, LNS, portfolio)
- Linear programming solver
- Graph algorithms (Max-Flow, Min-Cost-Flow, Assignment)

Mostly C++

- Bindings in Java, Python, and C#.

Large variety of uses!

Our problems are solved...

- In anywhere from few milliseconds to more than a day.
- Many users want solutions in < 5 minutes.
- Many users want solution *interactively* < 30 seconds.

To reduce problem solver time:

- Devise a custom technique (only for big projects)
- Decompose the problem (not always possible)
- Parallelize the solver itself (Google has capacity; why not use it?)

Consistency in solvers

Coming back to the initial title of the talk…

1. Optimization in Google
2. **Consistency in sequential solvers**
3. Consistency in parallel solvers

Introduction: Determinism/Consistency.

- If two solves on the same machine return different solutions:

  The solver is **non-deterministic**.

- If a solve with a fixed time limit ever outperforms a run with a *larger* time limit:

  The solver is **inconsistent**.

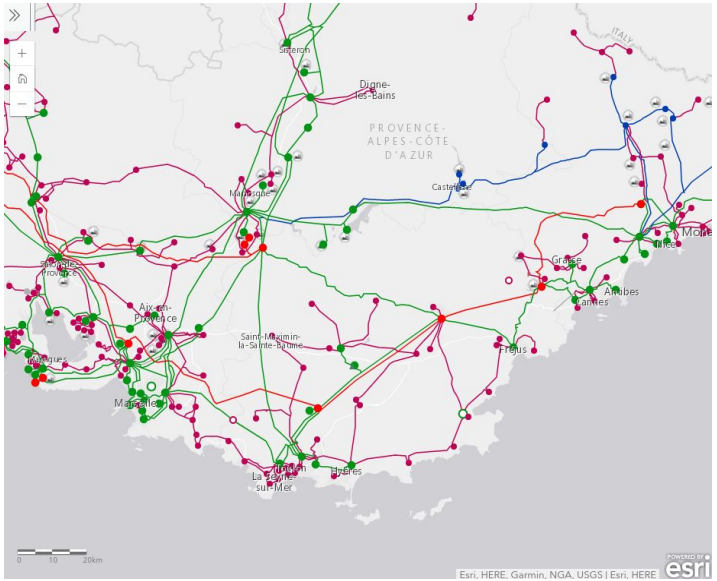
- If a parallel solve with a fixed number of cores ever outperforms a run with *more* cores:

  The solver is **inconsistent**.

An old example

The French agency “*Reseau de transport d’electricite*” (energy network) must be able to replay the MIP solver to prove to the control administration “*cours des comptes*” that the new high-power lines are the optimal ones.

Determinism is essential.



A recent example

Recent tests show that for a given problem instance, the solution computed by a commercial MIP solver in 5 minutes is superior to the solution computed in 15 minutes.

Why?

It is reasonable to think that according to the size of the problem and the time limit, the solver chooses quite different strategies. Perhaps:

- Focusing on heuristics to find a good solution when the time limit is small
- ...but attempting a complete search when the time limit is bigger.

Why are determinism and consistency important?

- Avoid disappointing users.
- Make debugging easier (or possible).
- Improve upon answers when more computing power is applied.
- Allow for interactive use.

Determinism or Consistency: Simple definition

When a solver always returns the same solution with the same context of execution:

the solver is deterministic

When a solver always returns a better or the same solution when the context of execution is “bigger”: the solver is consistent.

Context of execution is defined by:

- The time limit.
- The number of cores.

Context of execution is the surface allowed for a task.

Determinism in the execution environment.

Modern operating systems schedule the tasks (e.g. the solver) when they want.
User time depends on the load of the machine:

> The amount of work executed on a given time depends on the load of the machine.

Many side effects are involved:

- Context switches,
- Core switches,
- Cache misses,
- ...

We consider a deterministic time for the rest of the talk.

Consistency in sequential solver.

Quite easy...

How to have determinism (sequential solver)

- No randomization (with different seed),
- No learning over different runs, don’t use memory that can influence or change the “search”.

The work performed by the solver is always the same.

How to have consistency (sequential solver)

Only the time limit may change:

- The same property as the ones for determinism.
- Don’t change the heuristics strategy according to the time limit

The work performed in a time limit *t* must be included in the work performed in time limit *t’* when *t<t’*.

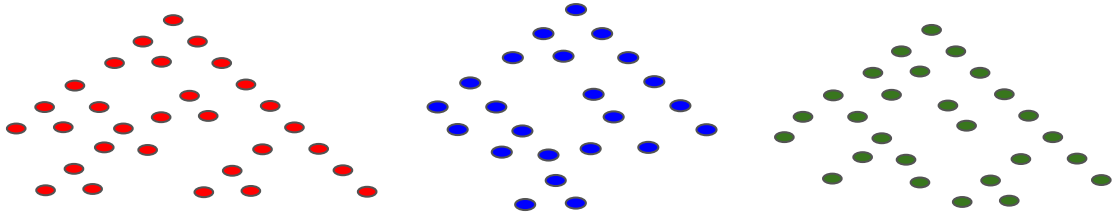
What about parallel solvers?

1. Portfolio parallel solver.
2. Parallel search.

Portfolio parallelization.

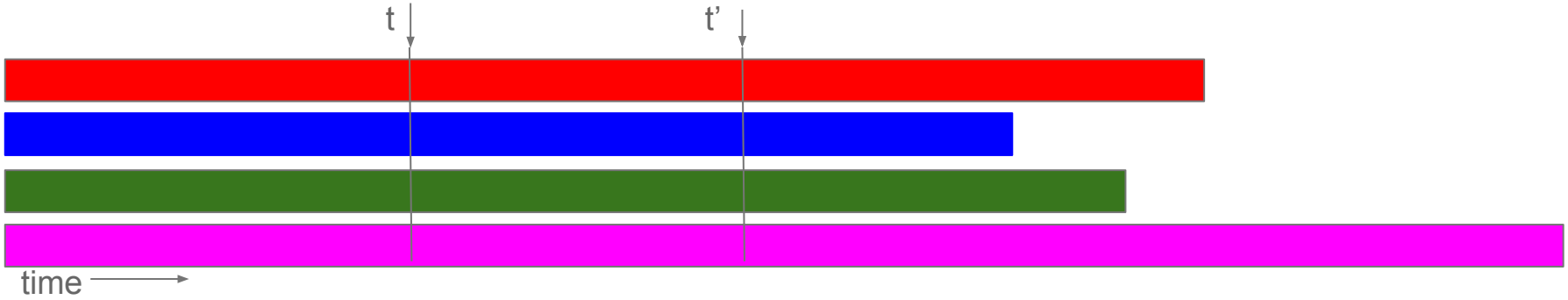
Several search space explorations:

- Different branching strategies
- Different search strategies



Independent searches:

- No communications until the end

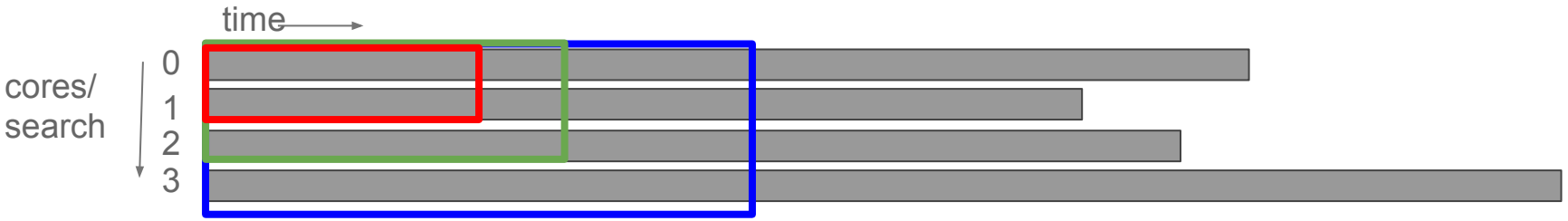


Portfolio parallelization

Determinism/Consistency without communication: Easy

- If each search is deterministic,
- If each search is statically defined

The work performed by a portfolio with *n* cores for a time limit *t* must be included in the work performed by a portfolio with *n'* cores for a time limit *t'* where *n<=n' and t<=t'*.

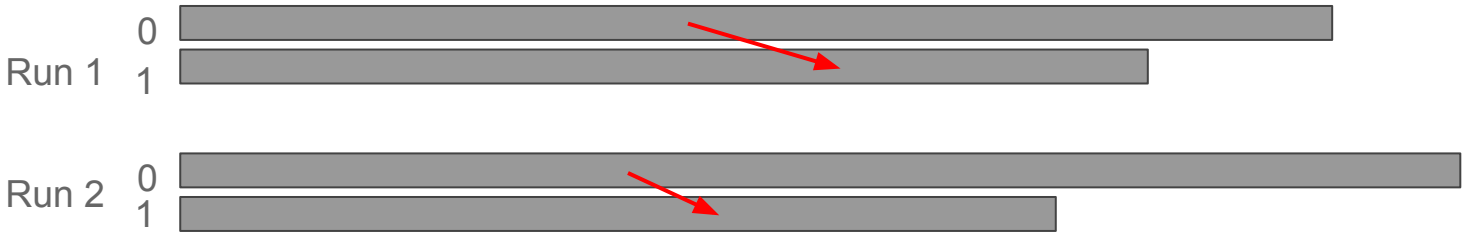


Portfolio parallelization with communication

Determinism depends on the determinism of the communication.

If the search S_i communicates something to S_{i+1}, what is the impact on the search S_{i+1}?

- One more time, there is no easy determinism for the execution of two threads on two cores,
- See determinism on communication on distributed system.



Portfolio parallel solvers

MIP: Gurobi offers portfolio parallel solver.

- multi-threaded and a distributed (MPI library)
- No mention of determinism.

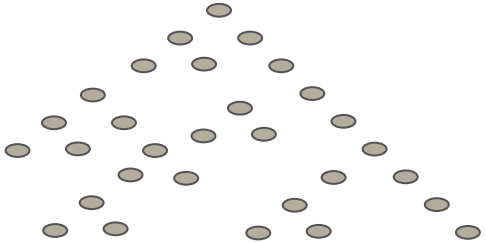
Typical for SAT or CP solvers (see Minizinc challenge).

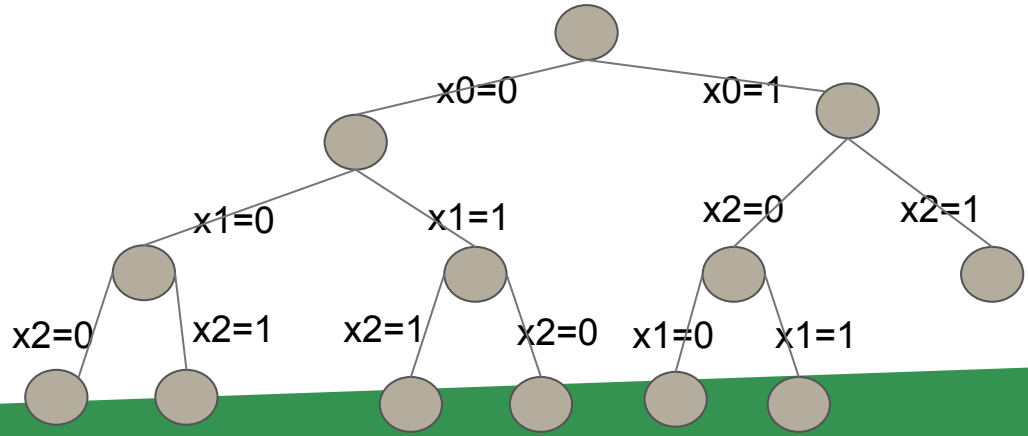
- Ppfolio, or-tools, Choco, IZplus, Gecode, Chuffed, ….

How to parallelize a solver? What does a solver do?

Abstraction of the sequential work of a solver:

- Exploration of the search space:
  - The search space is a graph or a tree.
  - The solver builds and explores each node of the search space.
- Shape of the search space is defined by branching strategies
- The scheduling of the search is defined by:
  - Search Strategies
  - DFS, BFS, Best First, Dive Search ...

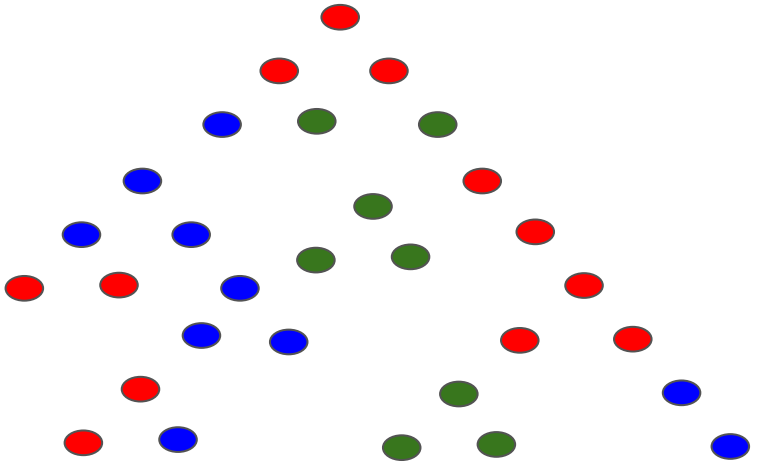




How to parallelize a solver?

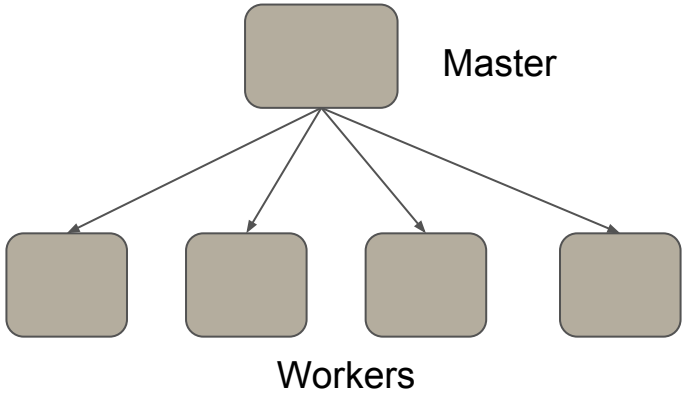
Assign parts of the search space to cores.

- Parallel search:
  - Dynamic mapping of nodes to computing resources.
- The search space is built on the fly:
  - Online allocation
  - Load balancing/work stealing
  - Online scheduling problem



Many implementations (MIP)

- Static tasks creation: pre-compute subtree generation:
  - Static allocation: round-robin, precompute the allocation of subtrees.
  - Dynamic allocation: get a subtree on demand.
- Dynamic tasks creation: creation of subtrees on demand.
  - Work-stealing,
  - Load balancing,
  - List algorithms.



Deterministic MIP

CPLEX implements a deterministic parallel solve, but no consistent parallel solve.

- Multi-threaded implementation.
- Each thread explores time-stamped batches of nodes.
- This batches management implies many synchronizations that reduce performance.
- No real determinism with time limit.

Possibility to switch to opportunistic parallel solver

- No determinism.
- More performance.

Parallel search: Is it difficult?

Another point of view: Parallel algorithms.

Data Flow graph (DFG):

- Tasks linked by data dependencies.

Scheduling Issues:

- What is the shape of the DFG? grid, tree, ...
- Is the DFG known at the beginning?
- Do we have to execute all the tasks?

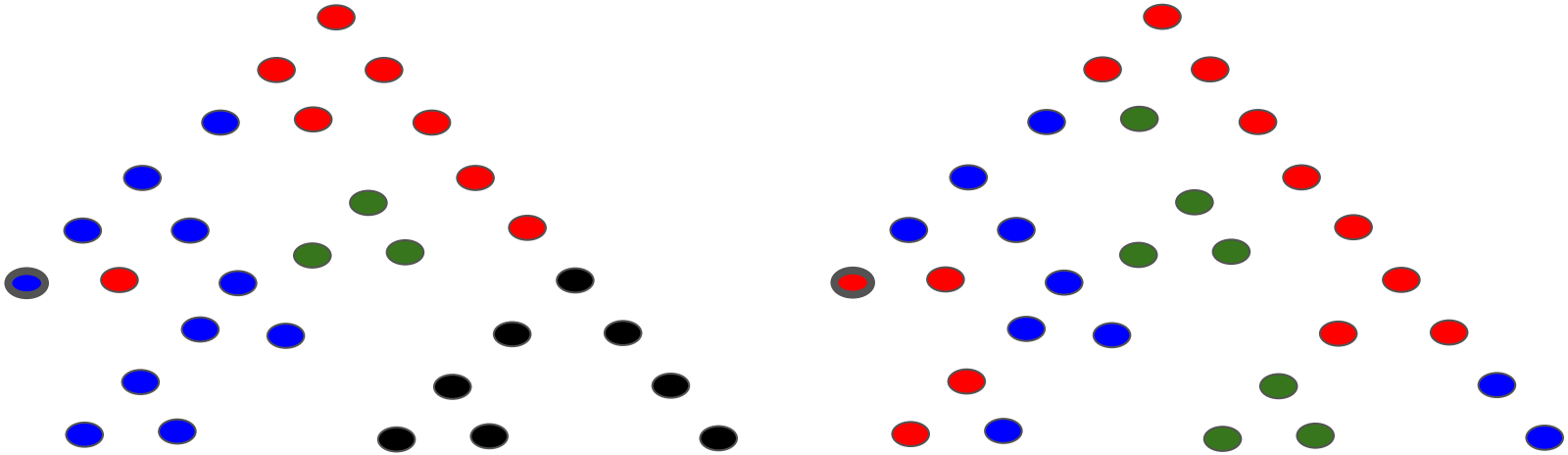
Taxonomy of parallel algorithms.

Parallel algorithms: taxonomy according to the DFG.

- DFG is known and is regular.
  - Example: sum of n values.
    simple static assignment: OpenMP.
- DFG is known and is not regular.
  - Example: sum of n values in sparse vector.
    static assignment, graph clustering: Metis, Scotch.
- DFG is unknown, but each tasks must be executed.
  - Examples: the previous one the tasks are created dynamically.
    online scheduling, list scheduling, work stealing: cilk, xkaapi, starPU.
- DFG is unknown, but number of tasks depends on the scheduling:
  - tree search, graph search.

Number of tasks vs scheduling

- Not usual in parallelism.
- Due to the pruning of search space.



Determinism/Consistency for parallel search.

- Need parallel solver.
- Need consistency.
- Need a formal explanation to prove what is possible and what is not.

In the following, I focus on MIP.

Hypothesis and notation:

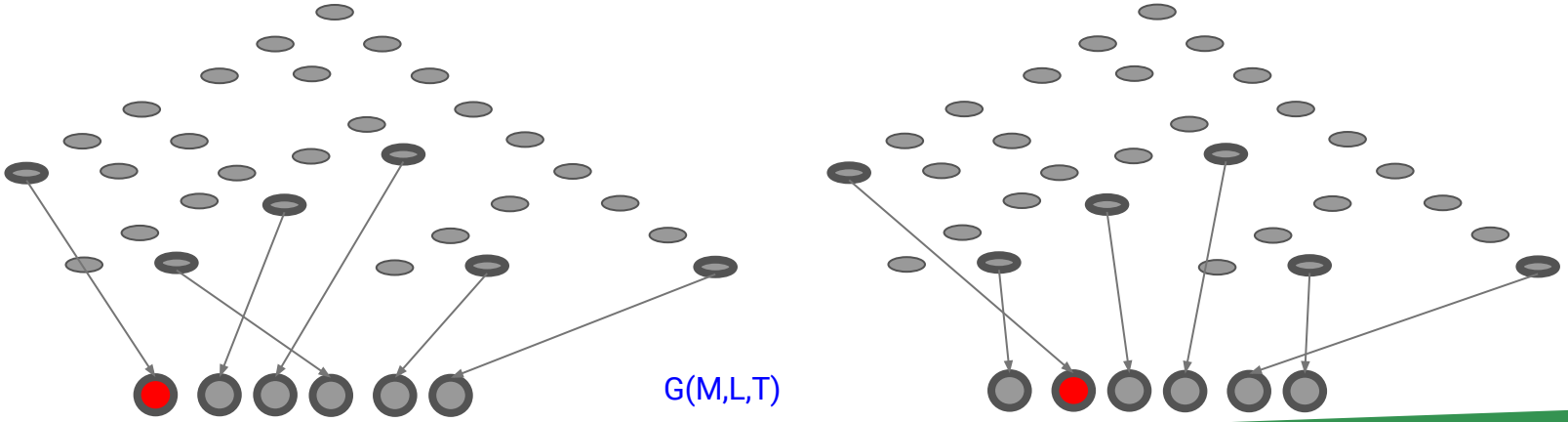
- Comparing two runs for the same problem.
- Deterministic time due to the non-exclusive access to the machines.
- Two different machines : impossible to compare, but if M1 is a subset of M2 : M2 > M1.
- Total order on solution value, if S1 = S2 equal solution value, but also equal variable assignments.
- Time limit noted L
- S=R(M,L,T): solving the problem on the machine M with time limit L at the time T gives the solution S.

(More) Formal definition of Consistency

- For all T1 and T2, R(M,L,T1)=R(M,L,T2): Two runs on the same machine with the same time limit must return the same solution.
- If M1<M2, R(M1,L,T1)<=R(M2,L,T2): A run on a given machine must return a solution better or equal as the solution returned by a run on a less powerful machine.
- If L1<L2, R(M,L1,T1)<=R(M,L2,T2): A run in a time limit L must returns a solution better or equal as the solution returned by a run with a smaller time limit.

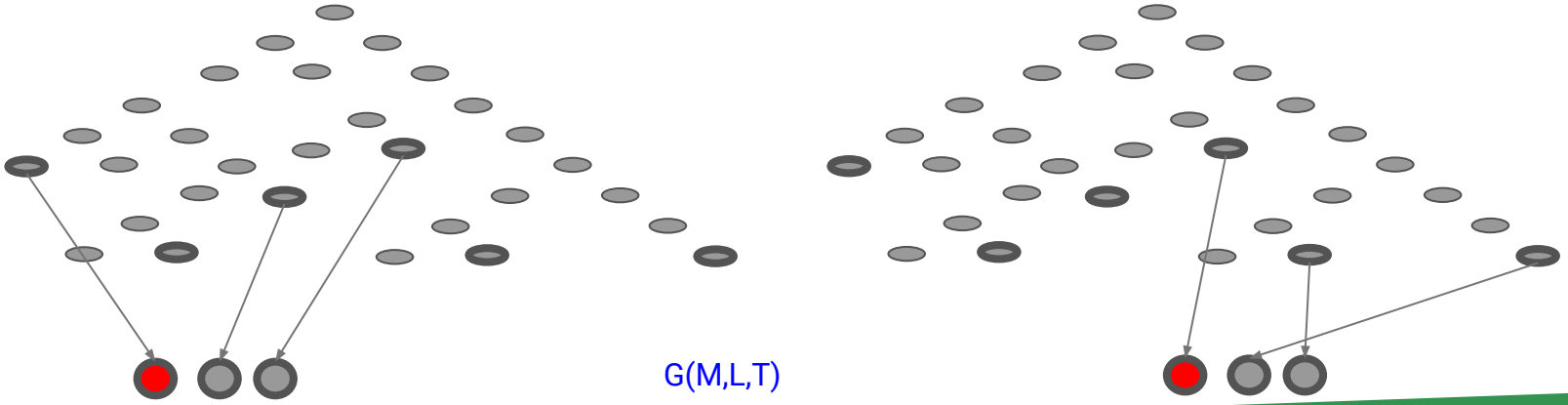
Implications for MIP

- R(M,L,T) defined as Sel(G(M,L,T)), where
  - G(M,L,T) returns the ordered list of solutions explored or discovered by a run of the solver on the machine M for a time limit L at a time T,
  - Sel() selects the solution from the ordered list of solutions found by the solver G().
- Discuss the consistency according to properties on G() and Sel()



No hypothesis on G(): solutions are returned in any order.

- If g1= G(M1,L1,T1) and g2=G(M2,L2,T2), two lists of returned solutions solving the same instance of problem but with different machine, time limit, start time.
- Clearly, g1 and g2 may have no common solution.
- But if L1 and L2 are equal to +infinity, we can ensure consistency if Sel() always returns the optimal solution with the minimal lexicographic order.



Hypothesis on G(): consistency

- Solutions are always found in the same order.
- Guarantee the list inclusion
- G() is consistent if it respects one of the following rules:
  - For all T1 and T2, G(M,L,T1)=G(M,L,T2).
  - If M1<M2, $G(M1,L,T1) \subseteq G(M2,L,T2)$.
  - If L1<L2, $G(M,L1,T1) \subseteq G(M,L2,T2)$.
  - if L1<L2 and M1<M2, $G(M1,L1,T1) \subseteq G(M2,L2,T2)$.

Not so easy to guarantee!

Conclusion for Parallel Search

- For one instance:
  - The tree must always be the same.
- Many implications:
  - No randomization,
  - Decision depends on the scheduling:
    - Restricted uses of learning
    - Pseudo costs
    - Difficult to use clause or constraint learning.

All the interesting features that reduce the search are forbidden!

Conclusion

- Industrial users are interested in determinism/consistency.
- Sequential solver
  - Determinism is easy...
  - ...and consistency is not too difficult.
- Portfolio solver
  - Determinism and consistency are not so hard.
  - Robust but no performance…
- Parallel Solver
  - Determinism is possible.
  - Consistency is not really possible without sacrificing performance.
  - Possible consistency for time limit only or machine only?

Another type of “determinism”

Iterative use of the solver. A solver is often used in a loop:

1. Solve the problem
2. Study the solution
3. Change the problem a bit to get a solution via a local modification.
4. Return to step 1.

When part of the problem is modified, users often want the rest of the solution to stay the same.

We often add a “solution distance” to the objective function to minimize the solution modifications…

Can we integrate this feature inside the solver as a “lazy” consistency ?

THANK YOU

Questions?

Hash Code