High-performance local search for planning maintenance of EDF nuclear park

Frédéric Gardi **Karim Nouioua**

Bouygues e-lab, Paris
fgardi@bouygues.com

Laboratoire d'Informatique Fondamentale - CNRS UMR 6166,
Faculté des Sciences de Luminy - Université Aix-Marseille II, Marseille
karim.nouioua@lif.univ-mrs.fr

EDF maintenance planning problem

Main features:

1) Scheduling outages: combinatorial decision variables, subject to hard constraints related to intervals on the integer line.

2) Planning production (stock refuels and power levels): continuous decision variables, subject to non linear flow/capacity constraints.

3) Uncertainty: modeled as scenarios for demands and T1 plants costs.

Originality arising in production planning: imposition constraints mixing continuous and discrete decisions.

→ Mixed integer non linear problem (MINLP)

EDF maintenance planning problem

Main difficulties:

1) Theoretical hardness: scheduling outages is NP-complete, production planning seems NP-hard too because of imposition constraints.

2) Very large-scale instances:

70 plants (T2) to schedule with 8 outages by plants over 300 weeks.

Production levels to plan for 170 plants (T1+T2) over 10000 time steps, according to 500 scenarios.

3) Standard computing resources: at most 1 hour of running time on a Linux x64 platform with 2.7 GHz core, 8 Go RAM, 6 Mo L2 cache.

Proposed practical solution:

Local Search

But…

1) Pure & direct : no decomposition, no hybridization, no metaheuristic.

2) Randomized: every decision made during the search is randomized.

3) Aggressive: millions of feasible solutions visited within the time limit.

Following the methodology by ESTELLON, GARDI, NOUIOUA [SLS 2009], derived from successful past experiences since 2004.

Methodology

Common vision:

Local Search = Metaheuristics

Our vision:

Local Search = Varied Moves + Incremental Evaluation

Experienced on very large-scale discrete problems with high economic stakes (but short running times):

- Car sequencing (Renault, 2005 Challenge*)

- Workforce and task scheduling (France Telecom, 2007 Challenge)

- Media planning (TF1 Publicité, 2011*)

Methodology

Extended to mixed-integer optimization:

- Inventory routing (Air Liquide, 2008*): MILP
- Resource scheduling for mass transportation (By TP, 2009*) : MILP
- Nuclear maintenance planning (EDF, 2010 Challenge): MINLP

Local Search is rarely used for mixed-integer optimization.

Main principle: combinatorial and continuous parts are treated together

→ Combinatorial and continuous decisions are simultaneously modified by a move during the search

Main difficulty: solving efficiently the continuous subproblem

Methodology

Work focused on

1) Designing moves enabling an effective exploration of search space.

2) Speeding up the evaluation of moves.

Here more particularly on

Implementing an incremental randomized combinatorial algorithm for solving approximately but very efficiently the continuous subproblem:

- 10 000 times faster than using LP (without imposition constraints)

- Near-optimal production plans

Methodology

Work surrounded by an important effort in software engineering for ensuring reliability of this critical evaluation machinery:

- programming with assertions
- checkers for incremental structures
- continuous refactoring
- CPU & memory profiling

→ Quest of high performance

Note : we have relaxed this effort the last week in order to concentrate our work on some improving technical features, and we have crashed…

General heuristic

Heuristic in 3 steps:

Step 1: Find an admissible scheduling of outages = respecting combinatorial constraints CT14-CT21.

Step 2: Find a schedule with admissible production plan = spacing outages such that stocks do not exceed maximum levels before and after refueling operations (CT11).

Step 3: Optimize the global cost of the admissible schedule.

Each step tackled by local search : first-improvement descent with randomized selection of moves.

Step 3: optimizing the global cost

How reducing the complexity induced by scenarios?

By working on a subset of scenarios (eventually aggregated).

The following strategy works well in practice:

3.1) Optimize on one scenario with average demands and T1 costs.
3.2) Refine the solution over all scenarios.

Step 3.1 is reinforced without losing efficiency: optimize on one scenario with average demands but with T1 completion costs computed over all scenarios.

Moves

Natural move:

Select *k* outages in the current solution and shift them over the time line. Size when no constraint: $O(H^k)$ with H the number of weeks.

Qualification stage: apply natural moves randomly with k = 1,2,3.

→ Very low success rate: premature rejection due to CT14-CT21

Idea: apply compound moves based on natural (small) moves, to reach feasible solutions with higher probability:

1) Apply a small move which may destroy feasibility.
2) Iterate small moves to repair violated combinatorial constraints.

Moves

Compound moves

→ Generalize ejection chains and destroy-repair methods
→ Jump from a feasible solution to another one by local search

Improvements:

- Select new starting dates respecting CT13 and CT11
- Target outages to destroy: random, constrained, consecutive
- Target outages to repair: inducing violations on CT14-CT21

→ 75 % of compound moves lead to new feasible solutions
→ Better convergence (speed, robustness)

Evaluation machinery

*** Methodological reminder [SLS 2009] ***

Local Search = incomplete search technique: its performance depends strongly on the number of solutions explored within the time limit.

evaluation machinery = high-performance algorithm engineering

1) Incremental algorithms relying on advanced data structures, exploiting invariants induced by moves → high-level efficiency

2) Careful implementation (cache-aware programming, CPU & memory profiling) → low-level efficiency

3) Programming with assertions, all data structures checked at each iteration in debug mode (checkers) → correctness & reliability

Evaluation machinery

General evaluation scheme, for a subset *S* of scenarios:

Combinatorial part:

Perform small destroying move ;
While combinatorial violations remain do
    Perform small repairing move ;
If solution remain infeasible, then abort ;

Continuous part:

Set refueling amounts of impacted outages ;
For each scenario in *S* do
    Set production levels of impacted T2 plants ;
Compute global cost of new solution ;

Evaluation machinery

Combinatorial part

Violations on CT14-CT21 maintained by $O(1)$-time routines related to the arithmetic of intervals (union, intersection, inclusion, distance).

Minimum Distance Cut:

Distance between consecutive outages k and $k+1$ must be large enough to ensure CT11 at $k+1$, even if fuel reload at k is minimal and production on cycle k is maximal.

→ Strong combinatorial "cuts" induced by the continuous sub problem

Evaluation machinery

Combinatorial part

Minimum Distance Cut:

- Evaluated in $O(\log T')$ time by dichotomy in the worst case
- But in $O(1)$ amortized time by hash-map caching in practice

T' : time steps between two consecutive outages

Since 80 % of the evaluation time is spent in the continuous part, then Minimum Distance Cut is crucial for efficiency.

Continuous part (for a given scenario)

For impacted outage *k*, refueling amount is randomly set between:

- the minimum given in input
- the maximum to satisfy the minimum spacing to outage *k+1*

Evaluation machinery

Continuous part (for a given scenario)

For impacted production cycle *k*, production levels are computed using an $O(T')$-time randomized-greedy algorithm:

1) Push production levels to maximum while imposition is not reached.

2) Compute analytically the maximum amount m^* of modulation.

3) Set modulation amount m randomly in $[0, m^*]$.

4) Stock s to consume = stock after refueling – m.

5) Set production levels so as to consume stock s: <u>either randomly from left to right, or driven by the lowest T1 completion costs</u>.

Evaluation machinery

Continuous part

For each impacted T2 plant, T1 completion costs are computed over all scenarios in $O(\log(P1\ S))$ time using an extensive data structure.

Total time complexity: $O(P2'\ T'\ (S + \log(P1\ S)))$

$P1$: T1 plants, $P2'$: impacted T2 plants,
T' : impacted time steps, S : scenarios.

→ Almost linear in the size of changes on current solution
→ 10 000 times faster than linear programming (Gurobi, CPLEX)
→ Near-optimal production plans (gap lower than 0.1 %)

Numerical experiments

Summary in numbers…

- Programmed in ISO C++: 12 000 lines of code
- 15 % of code dedicated to checks (3 debug levels)
- Statically compiled with GCC 4 (-O3) on x86-64 platform
- *Gprof* for CPU profiling, *Valgrind* for memory profiling

- Continuous part treated in exact precision using 64-bits integers
- Low-level code optimization to reduce RAM footprint by 2
- 1.7 GB of RAM allocated for largest instances (B8-10, X13-15)

- Fast convergence: 99 % of cost improvement in 10 minutes
- 20 000 compound moves per minute (100 000 small moves)
- 1 000 000 of feasible solutions explored per hour
- Improvement rate of compound moves: 1 %

Numerical experiments

Results obtained on 64-bits Linux with 2.93 GHz, RAM 4 GB, L2 4 MB (no parallelization).

Instances	10 minutes	1 hour	10 hours	Best	Gap
A01	1.694 804 e11	1.694 780 e11	1.694 748 e11	1.695 383 e11	- 0.036 %
A02	1.459 699 e11	1.459 600 e11	1.459 568 e11	1.460 484 e11	- 0.061 %
A03	1.543 227 e11	1.543 212 e11	1.543 160 e11	1.544 298 e11	- 0.070 %
A04	1.115 163 e11	1.114 966 e11	1.114 940 e11	1.115 913 e11	- 0.085 %
A05	1.245 784 e11	1.245 599 e11	1.245 439 e11	1.258 222 e11	- 0.989 %

Remark : modulation enables to gain nearly 1 % on instance A05.

Numerical experiments

Once the bug corrected, we obtain the following results on instances B:

Instances	10 minutes	1 hour	10 hours	Best	Gap
B06	8.413 041 e10	8.387 786 e10	8.379 878 e10	8.342 471 e10	+ 0.543 %
B07	8.118 554 e10	8.117 563 e10	8.109 972 e10	8.129 041 e10	- 0.129 %
B08	8.241 156 e10	8.196 477 e10	8.189 974 e10	8.192 620 e10	+ 0.047 %
B09	8.219 982 e10	8.175 367 e10	8.168 956 e10	8.261 495 e10	- 1.043 %
B10	7.805 363 e10	7.803 998 e10	7.791 096 e10	7.776 702 e10	+ 0.351 %

Global average gap (used to rank competitors):

- Greater than 1 % with solution approach ranked 3rd

- Greater than 10 % with solution approach ranked 6th

Numerical experiments

Once the bug corrected, we obtain the following results on instances X:

Instances	10 minutes	1 hour	10 hours	Best	Gap
X11	7.919 385 e10	7.910 063 e10	7.900 765 e10	7.911 677 e10	- 0.020 %
X12	7.760 939 e10	7.760 090 e10	7.756 399 e10	7.763 413 e10	-0.043 %
X13	7.652 986 e10	7.637 339 e10	7.628 852 e10	7.644 920 e10	- 0.099 %
X14	7.631 402 e10	7.615 824 e10	7.614 948 e10	7.617 299 e10	-0.019 %
X15	7.444 765 e10	7.439 302 e10	7.438 837 e10	7.510 139 e10	-0.943 %

Global average gap (used to rank competitors):

- Greater than 1 % with solution approach ranked 3rd

- Greater than 10 % with solution approach ranked 6th

References

For more details on local search for mixed-integer optimization:

T. BENOIST, B. ESTELLON, F. GARDI, A. JEANJEAN (2011). Randomized local search for real-life inventory routing. To appear in *Transportation Science*.

F. GARDI, K. NOUIOUA (2011). Local search for mixed-integer nonlinear optimization: a methodology and an application. To appear in *Proceedings of EvoCOP 2011*, *Lecture Notes in Computer Science*. Springer.

Web : http://pageperso.lif.univ-mrs.fr/~frederic.gardi

LocalSolver

Based on these past experiences, we start developing in 2007 a black-box solver based on local search for combinatorial optimization.

Bouygues e-lab: T. BENOIST, F. GARDI, R. MEGEL
LIF - Université Aix-Marseille: B. ESTELLON, K. NOUIOUA

LocalSolver 1.x is able to tackle large-scale 0-1 (nonlinear) programs.

The software can be downloaded and used freely at:

www.localsolver.com