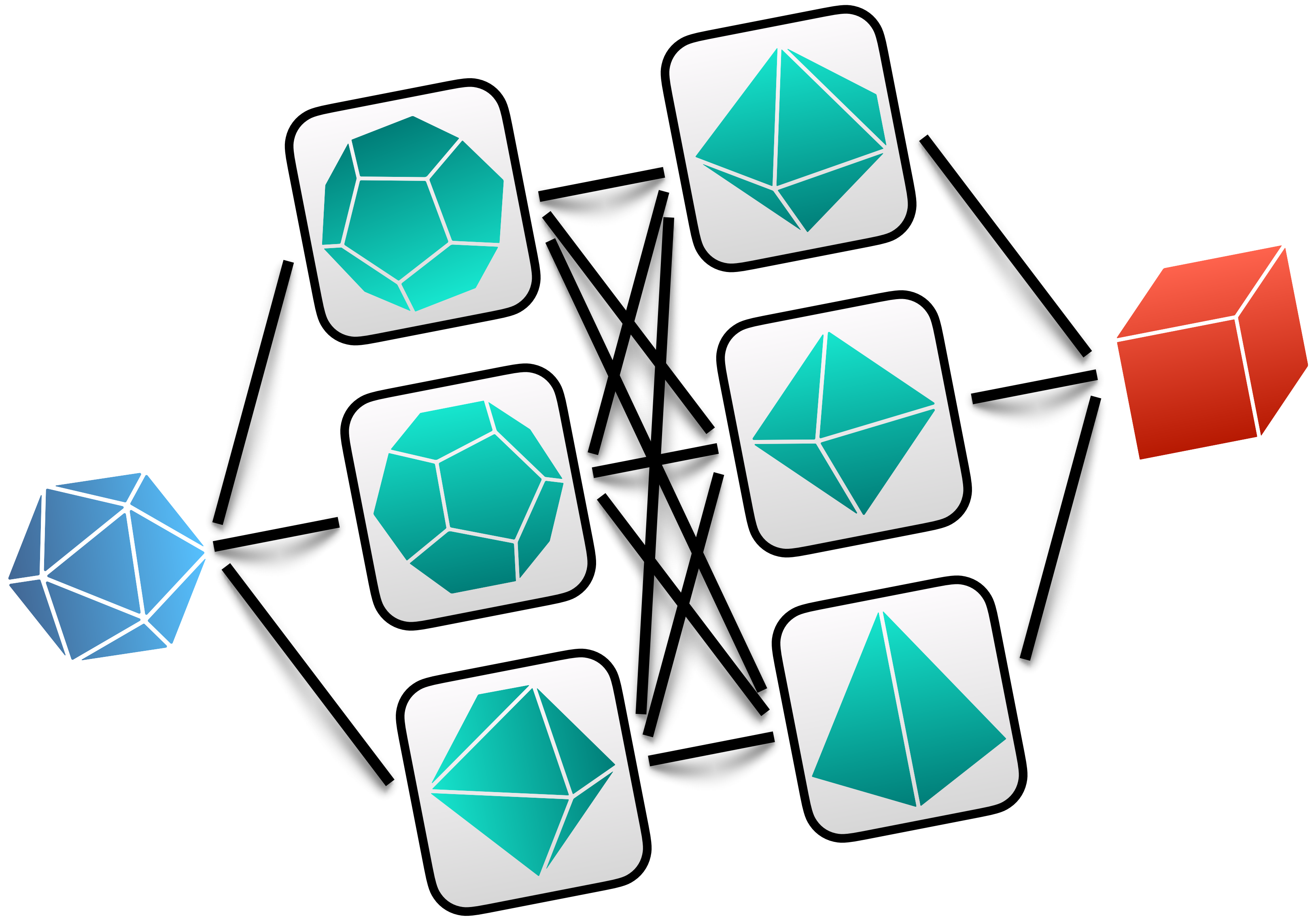
Did I forget to hit record? Please remind me!



Algorithm Configuration and friends

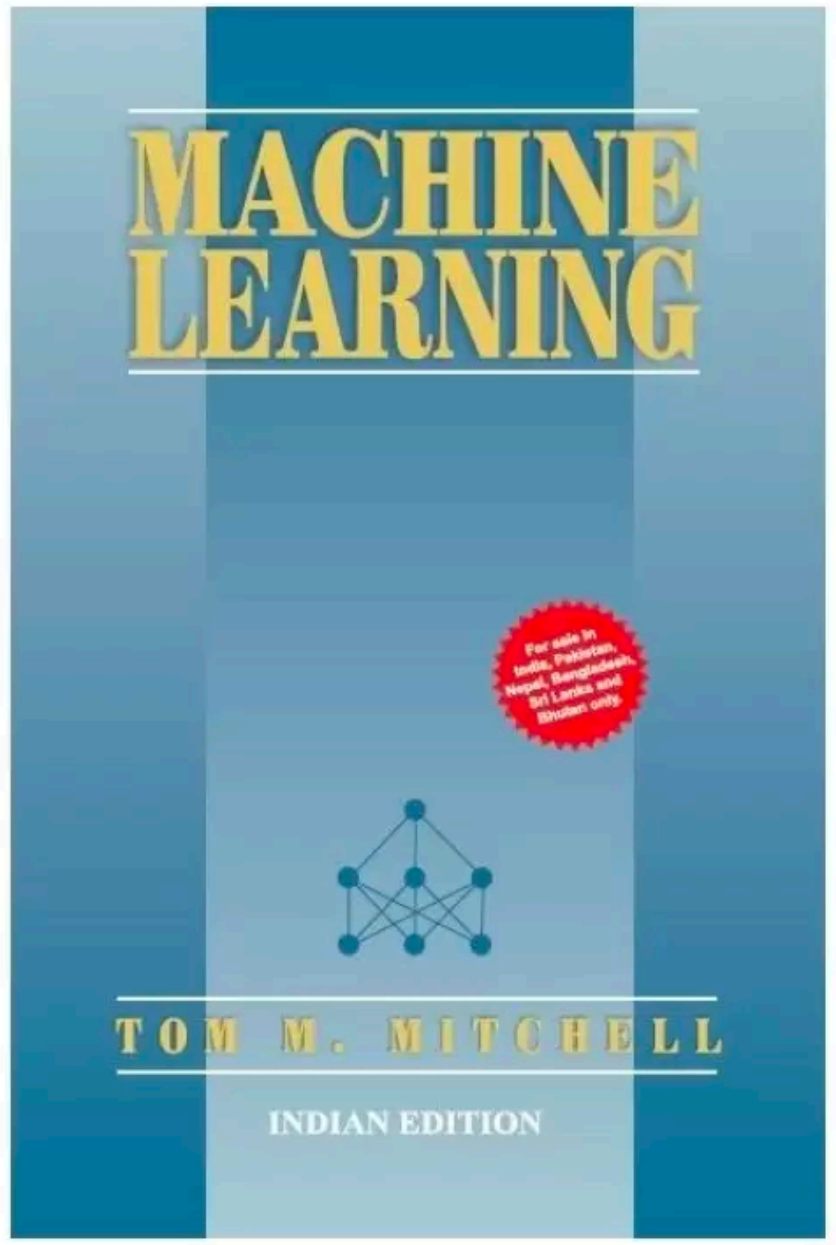
MIE1666: Machine Learning for Mathematical Optimization

Largely based on Hoos, Holger H. "Automated algorithm configuration and parameter tuning." Autonomous search. Springer, Berlin, Heidelberg, 2011. 37-71.

Some examples from Chapter 13 of Integer Programming by Wolsey

Elias B. Khalil — 04/10/21

UNIVERSITY OF TORONTO



Machine Learning:

Study of algorithms that

- improve their <u>performance</u> P
- at some <u>task</u> T
- with <u>experience</u> E

well-defined learning task: <P,T,E>

Tom Mitchell, CMU 10-601 slides

Why should this work at all?

The main theoretical basis of supervised learning:

With a **sufficient amount of "similar" data**

+

an **expressive model class**:

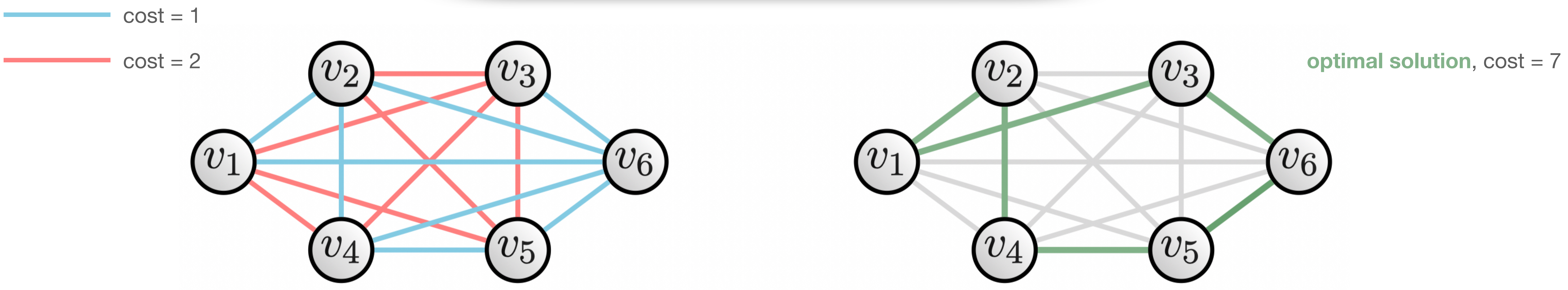
Minimizing the loss function on the training data yields a highly accurate model on **unseen test data**, with high probability

1. **Data**: $S = \{(x_i, y_i)\}_{i = 1,\ldots,n}$
   - x_i: data example with d attributes
   - y_i: label of example (what you care about)

2. Classification **model**: a function $f_{(a,b,c,\ldots)}$
   - Maps from X to Y
   - (a,b,c,…) are the **parameters**

3. **Loss** function: **L(y, f(x))**
   - Penalizes the model's mistakes

Graph Optimization

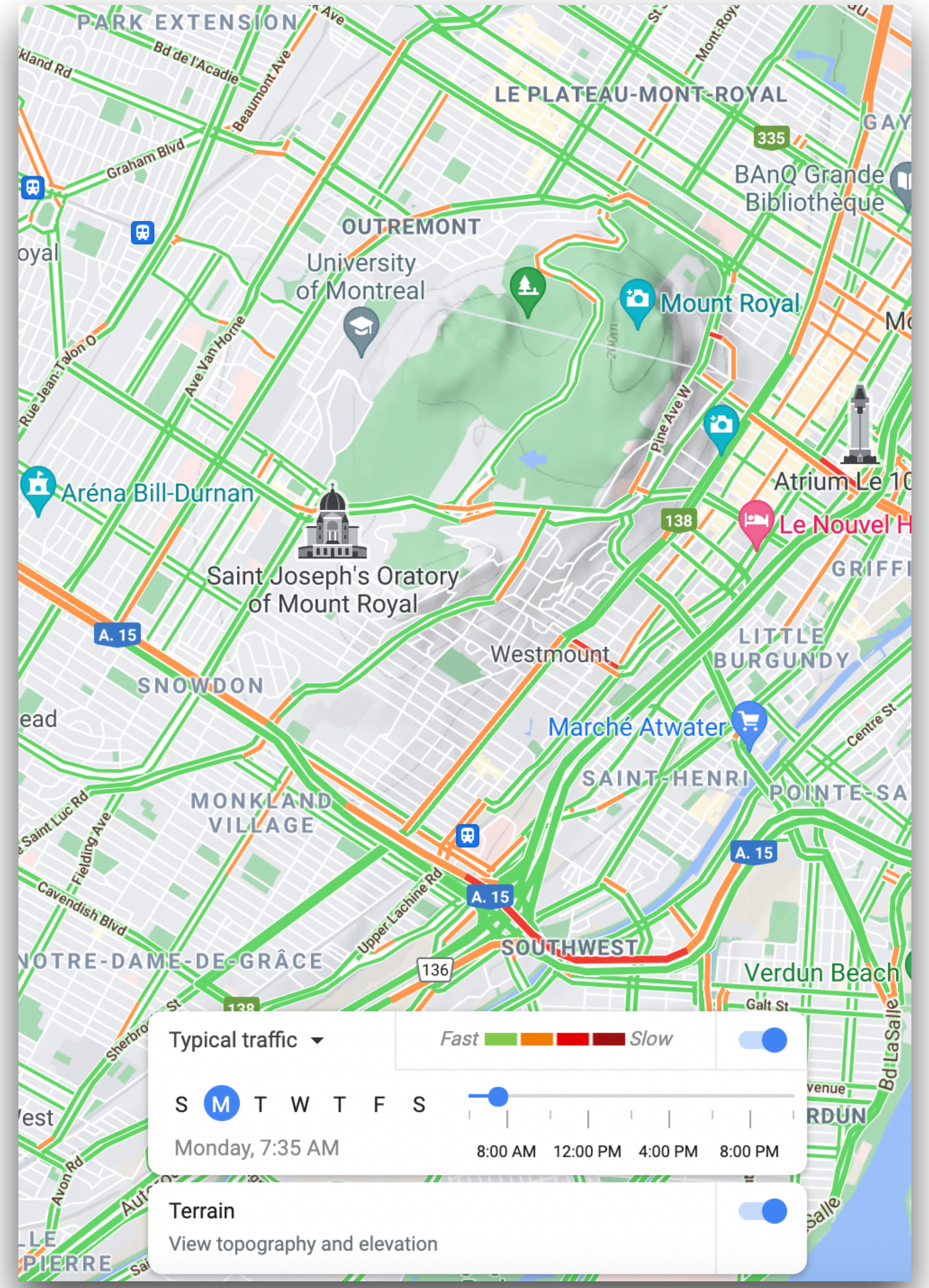


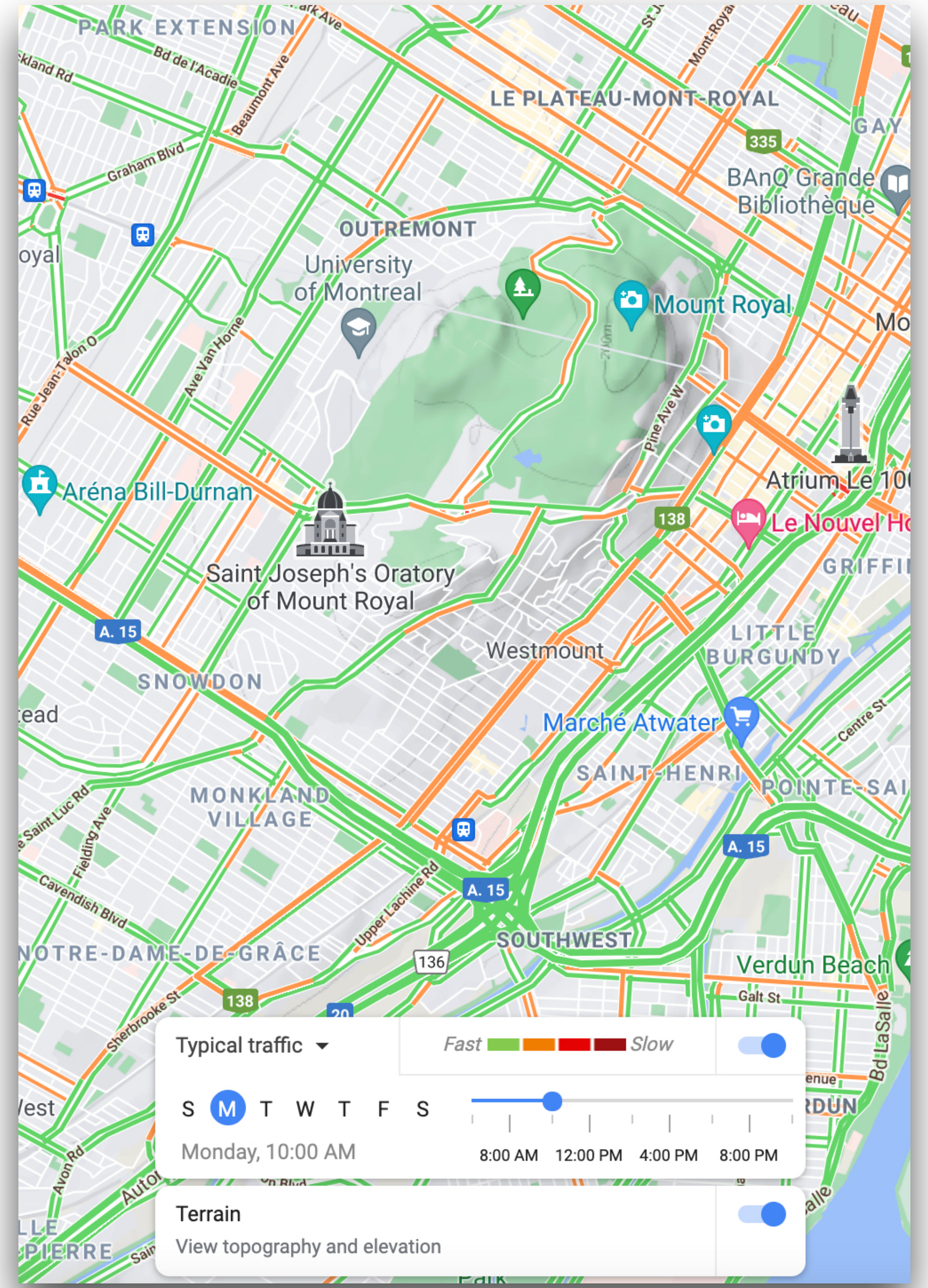
Travelling Salesperson Problem (TSP)

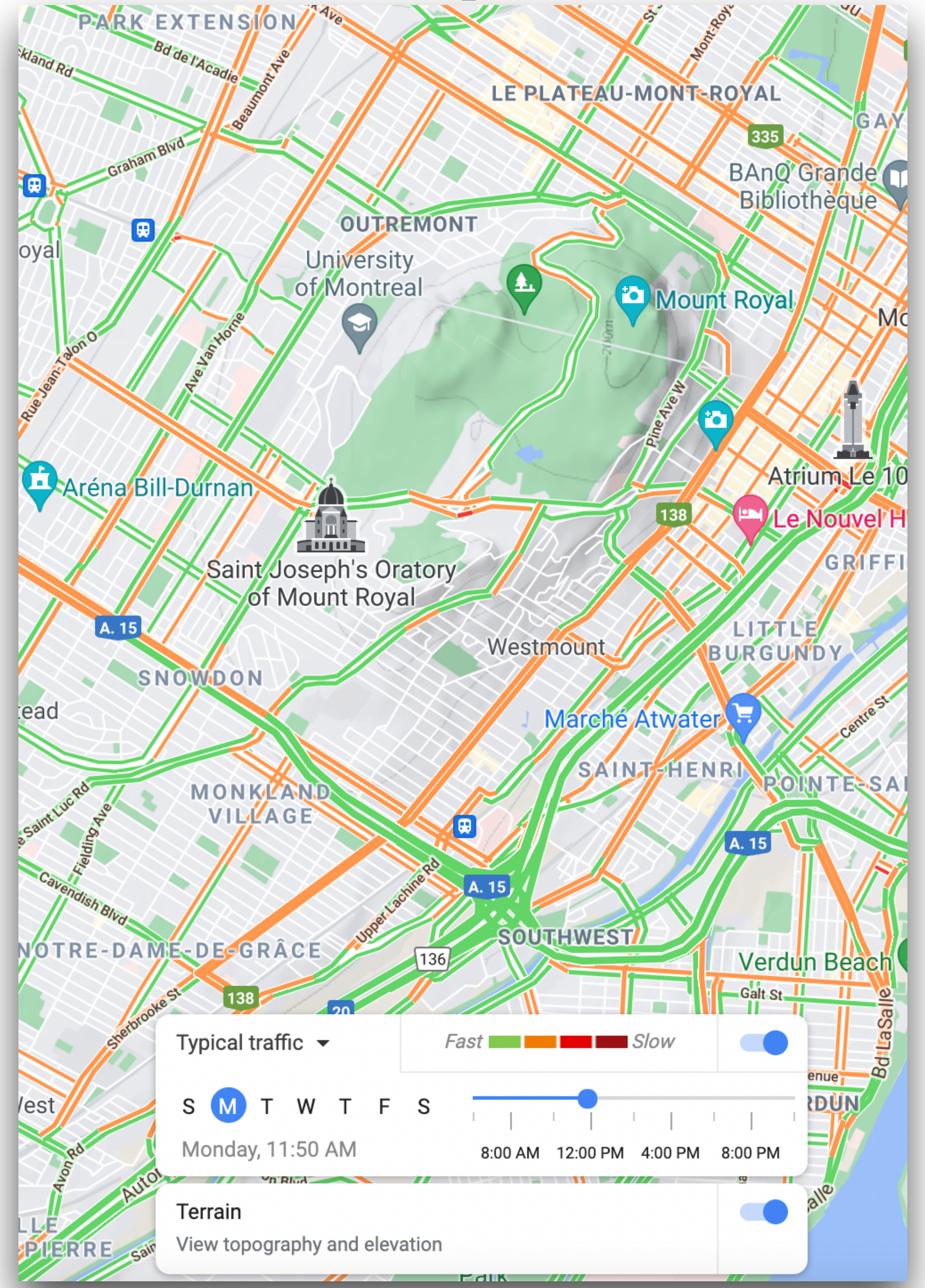
TSPs in Montréal

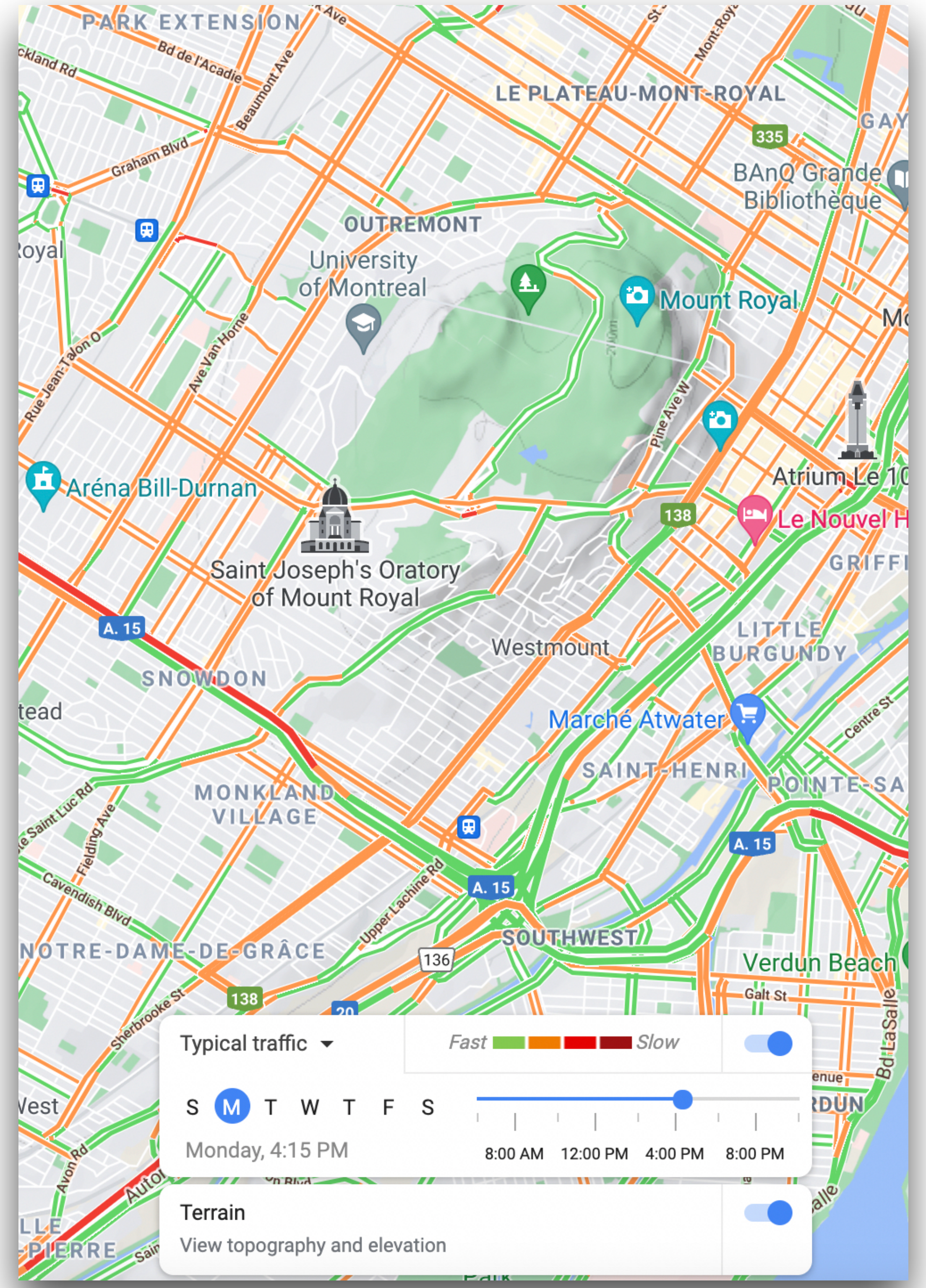
Every few hours, optimally **route** the salesperson through a set of locations

8 am	10 am	12 pm	4 pm







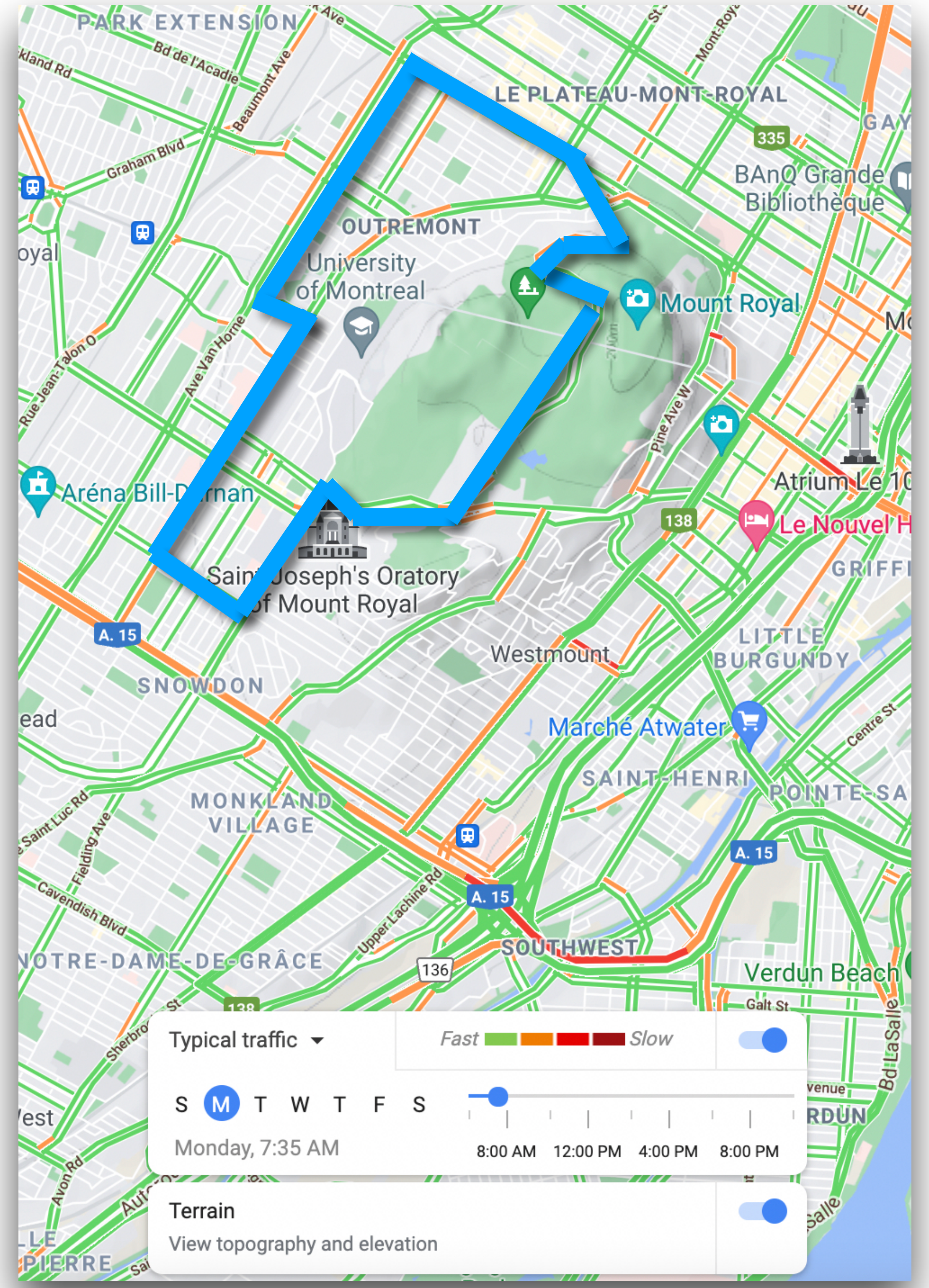


Google Maps, Montréal, Quebéc, Canada

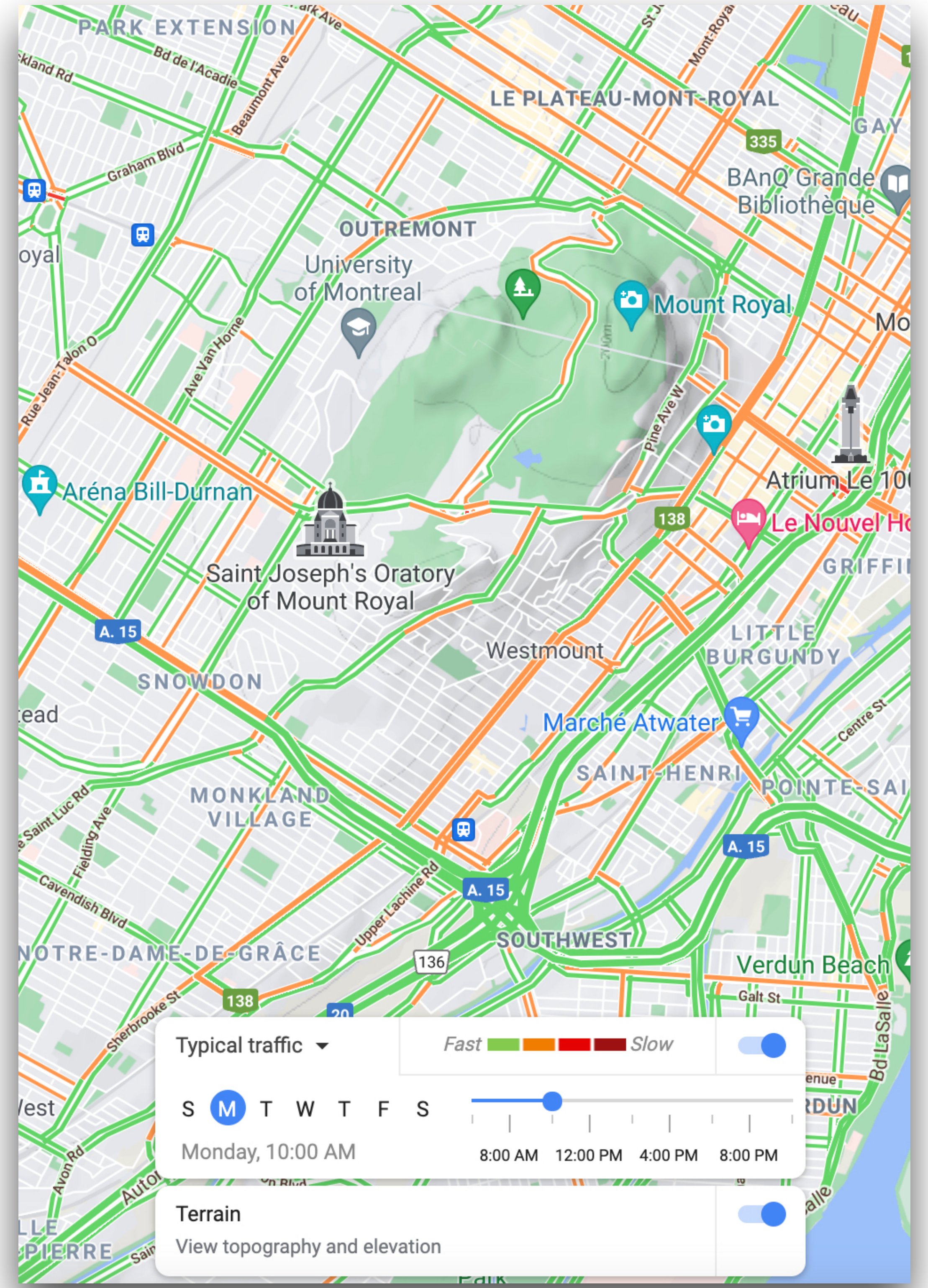
TSPs in Montréal

Every few hours, optimally **route** the salesperson through a set of locations

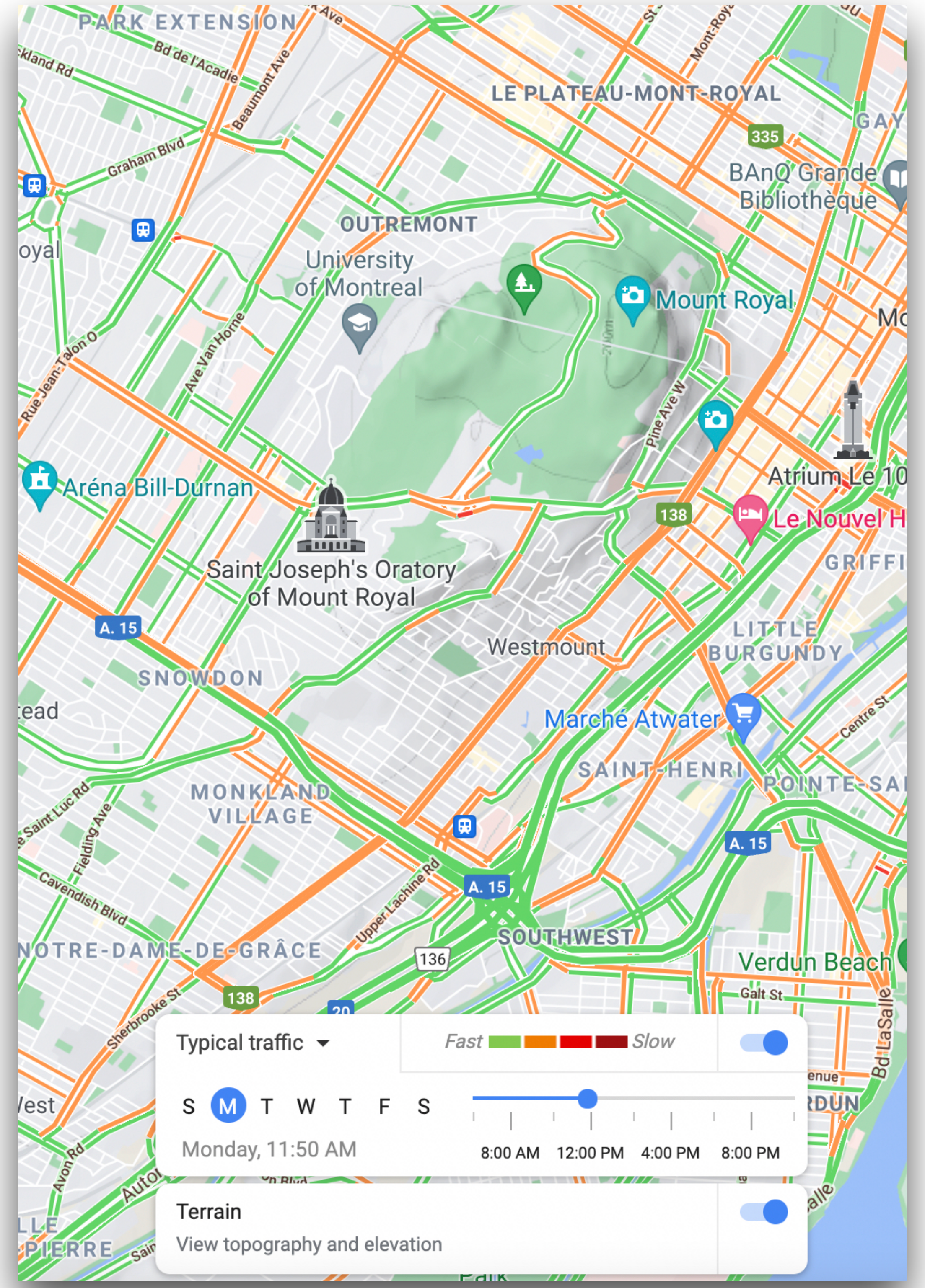
8 am



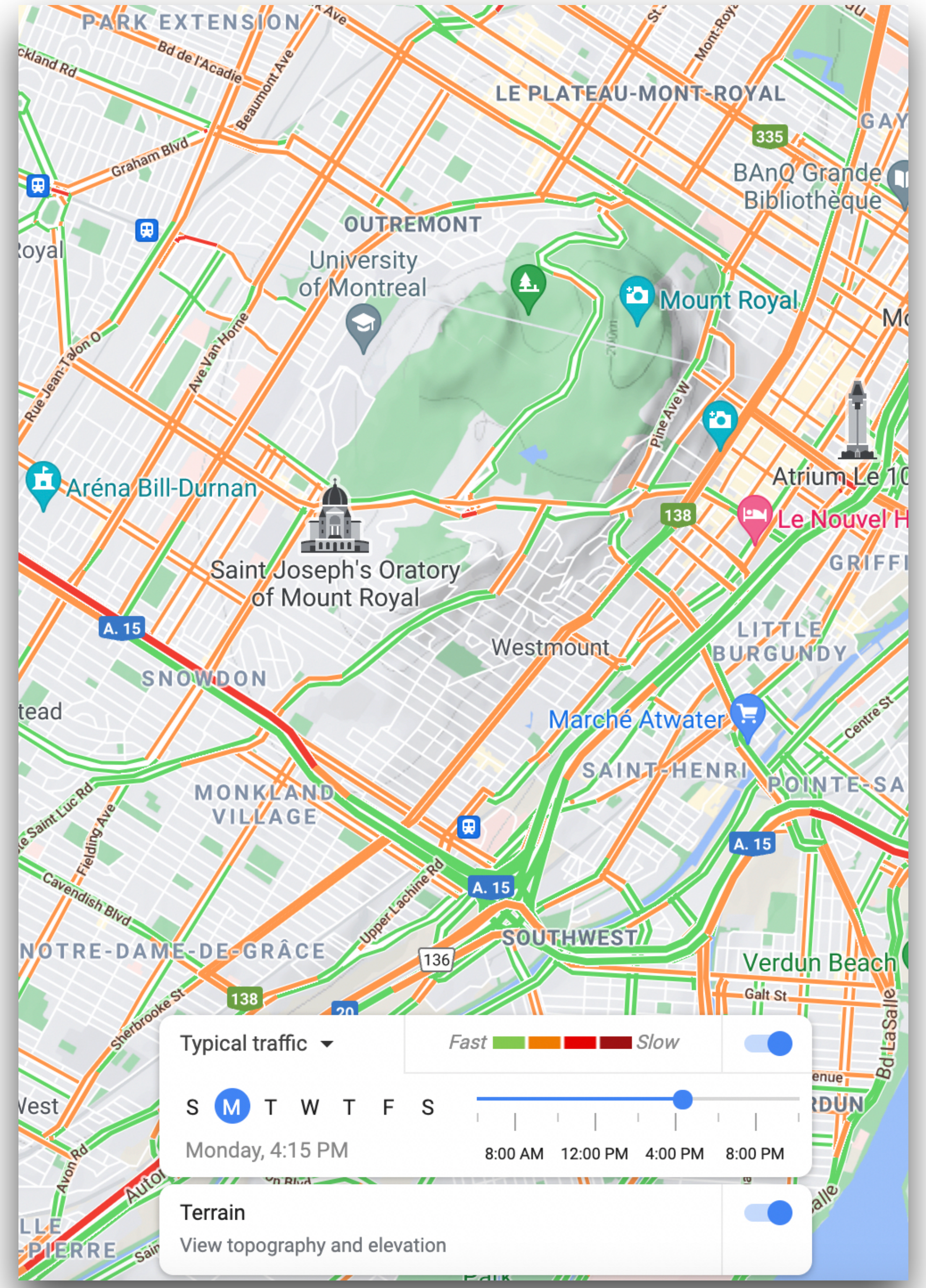
10 am



12 pm



4 pm

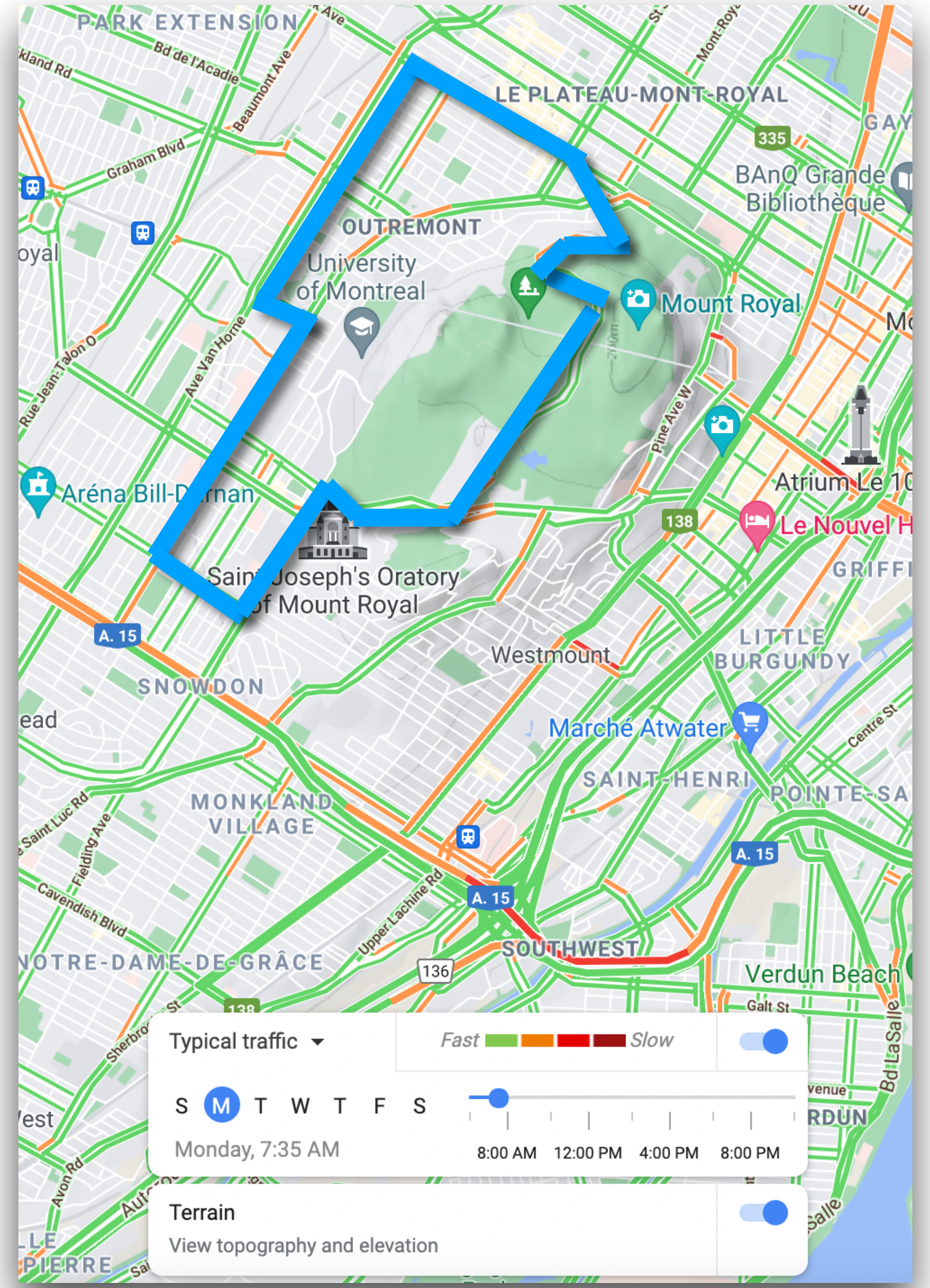


Google Maps, Montréal, Quebéc, Canada

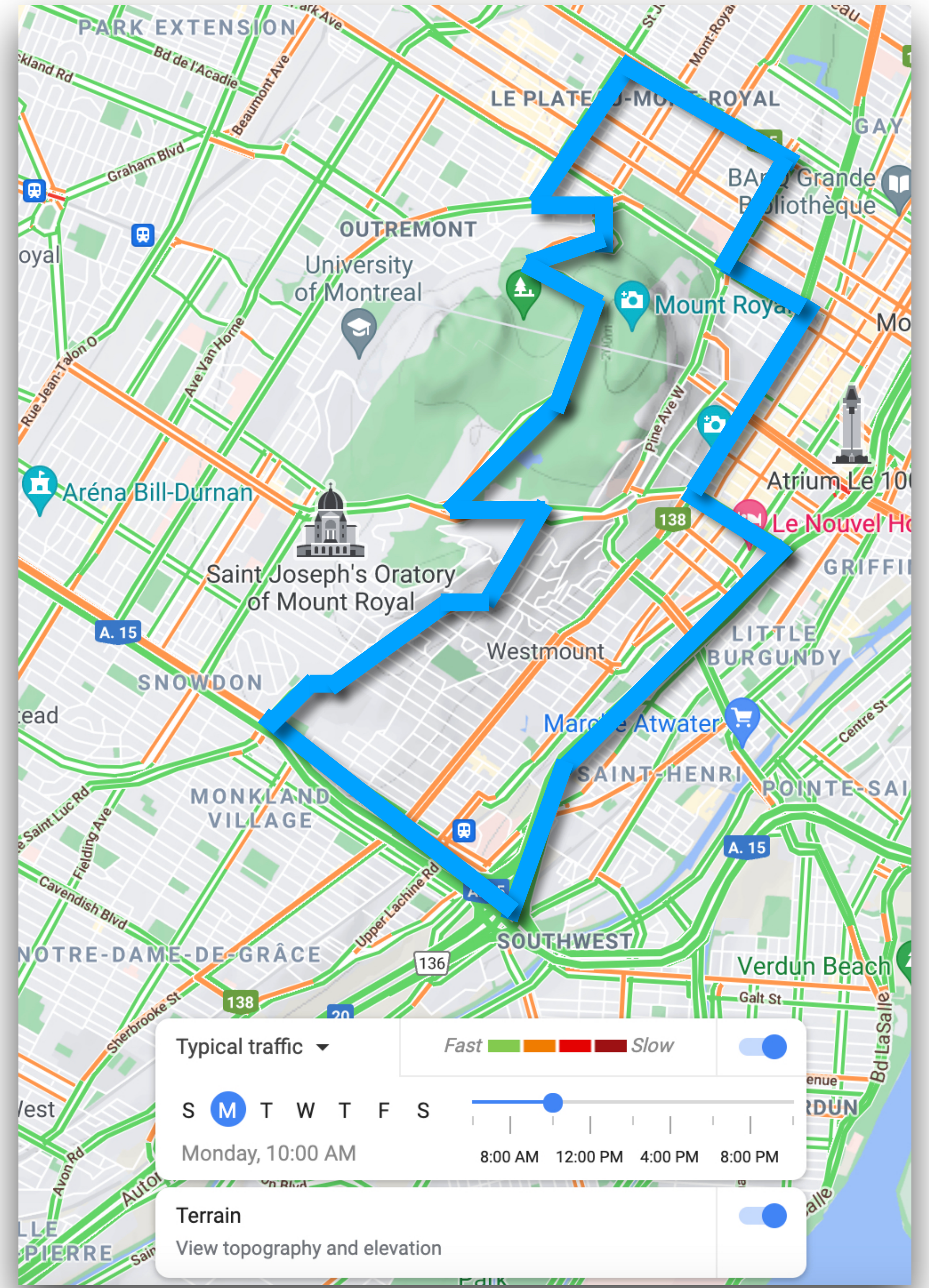
TSPs in Montréal

Every few hours, optimally **route** the salesperson through a set of locations

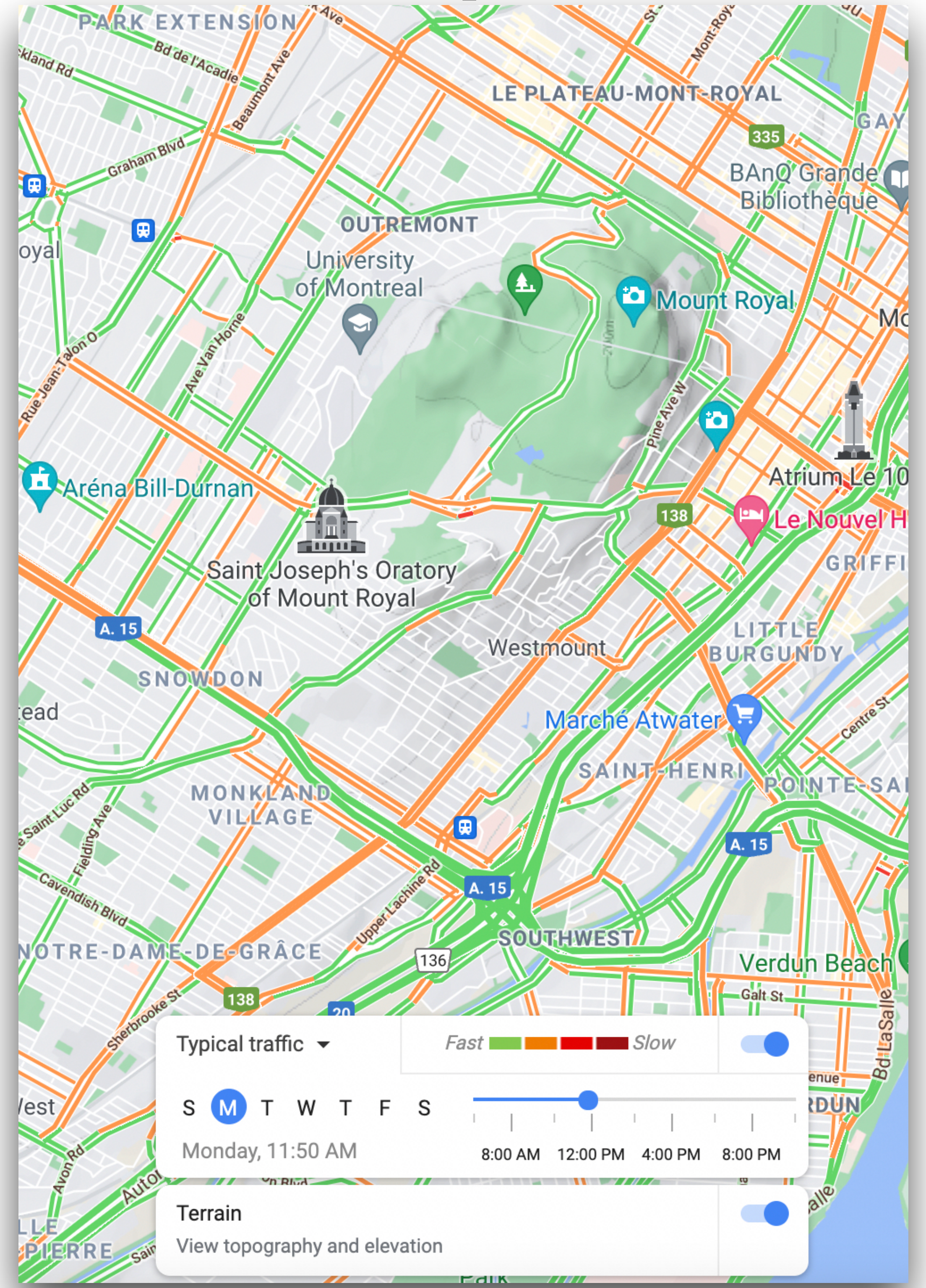
8 am



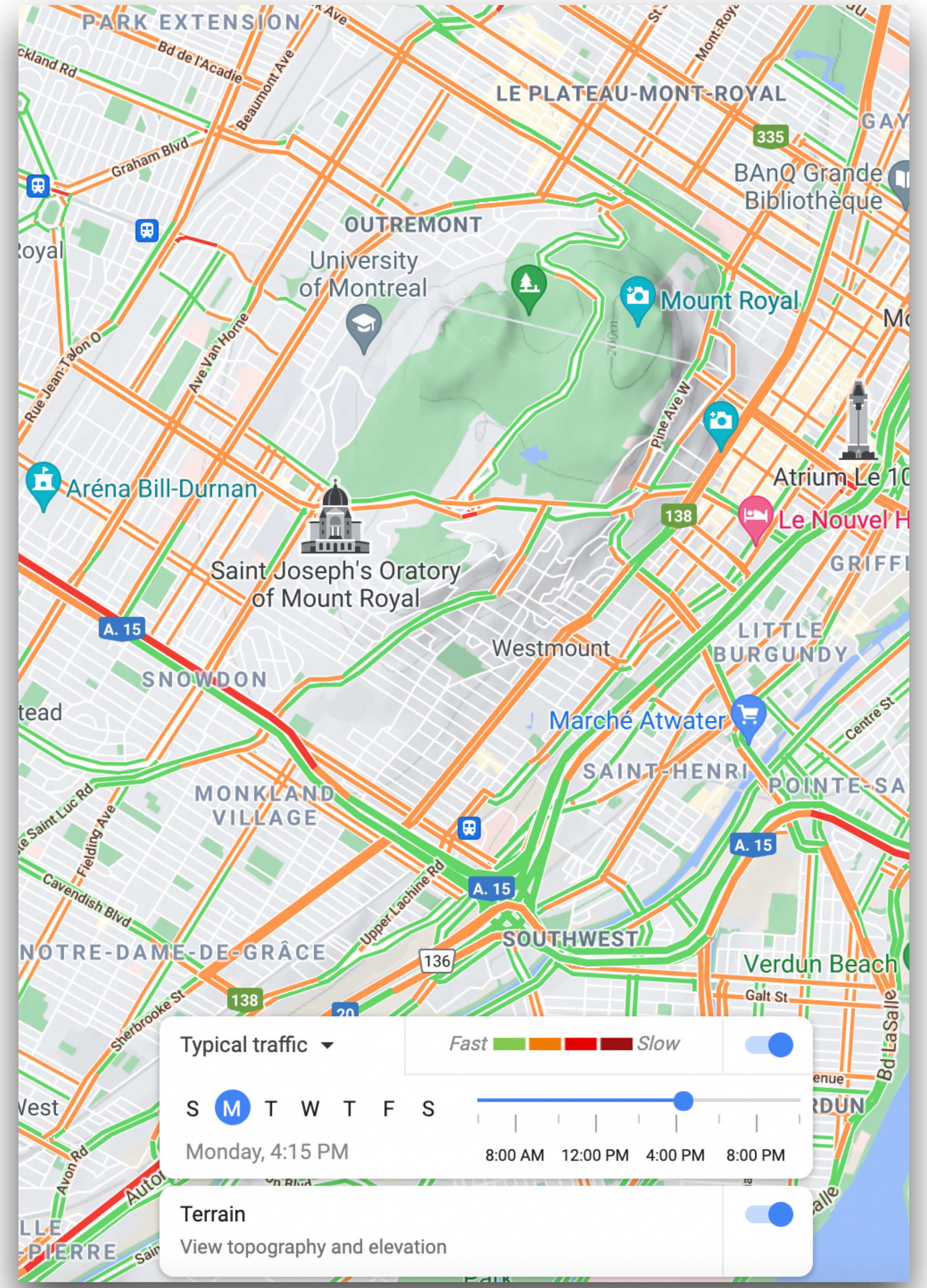
10 am



12 pm



4 pm

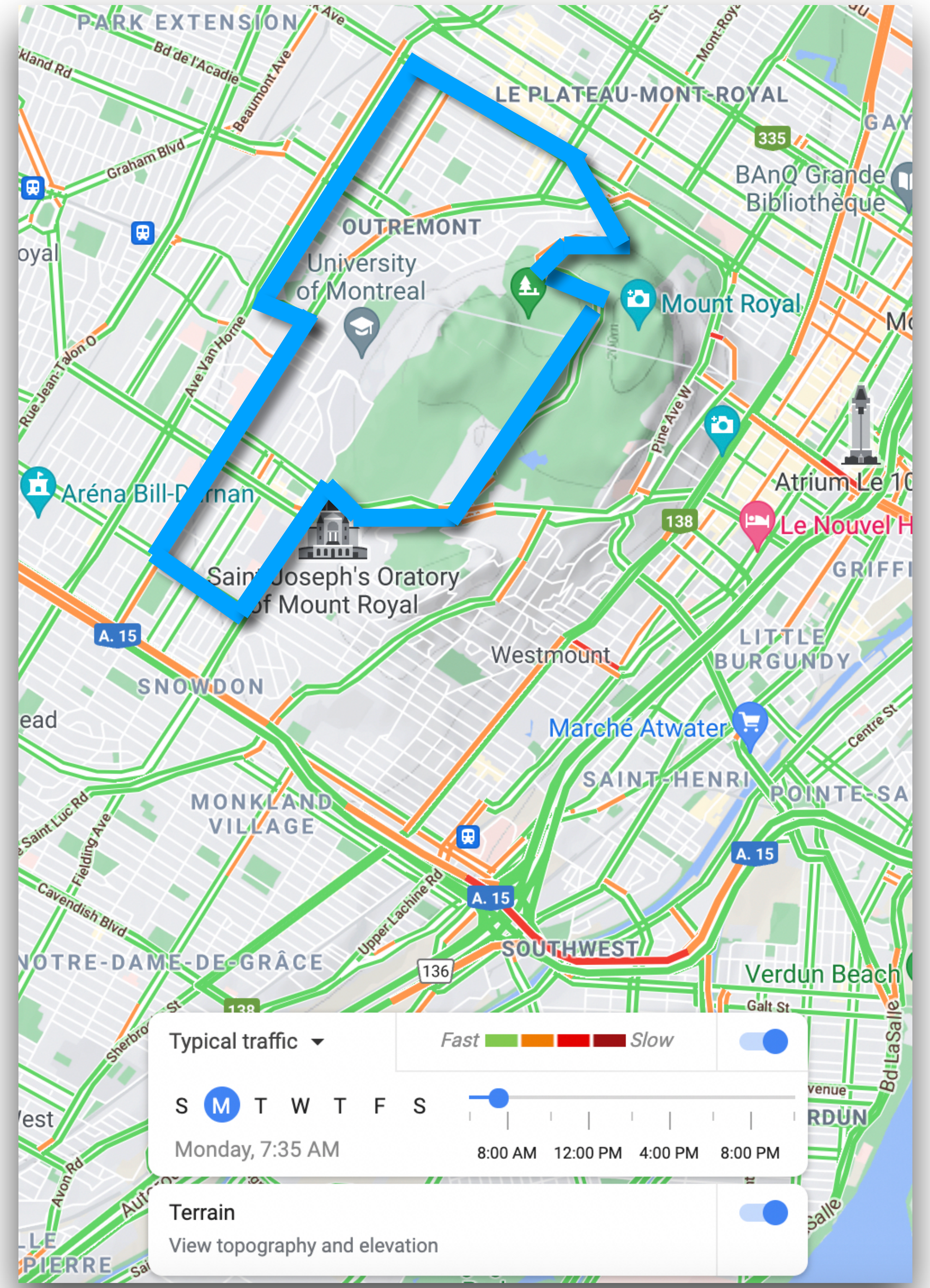


Google Maps, Montréal, Quebéc, Canada

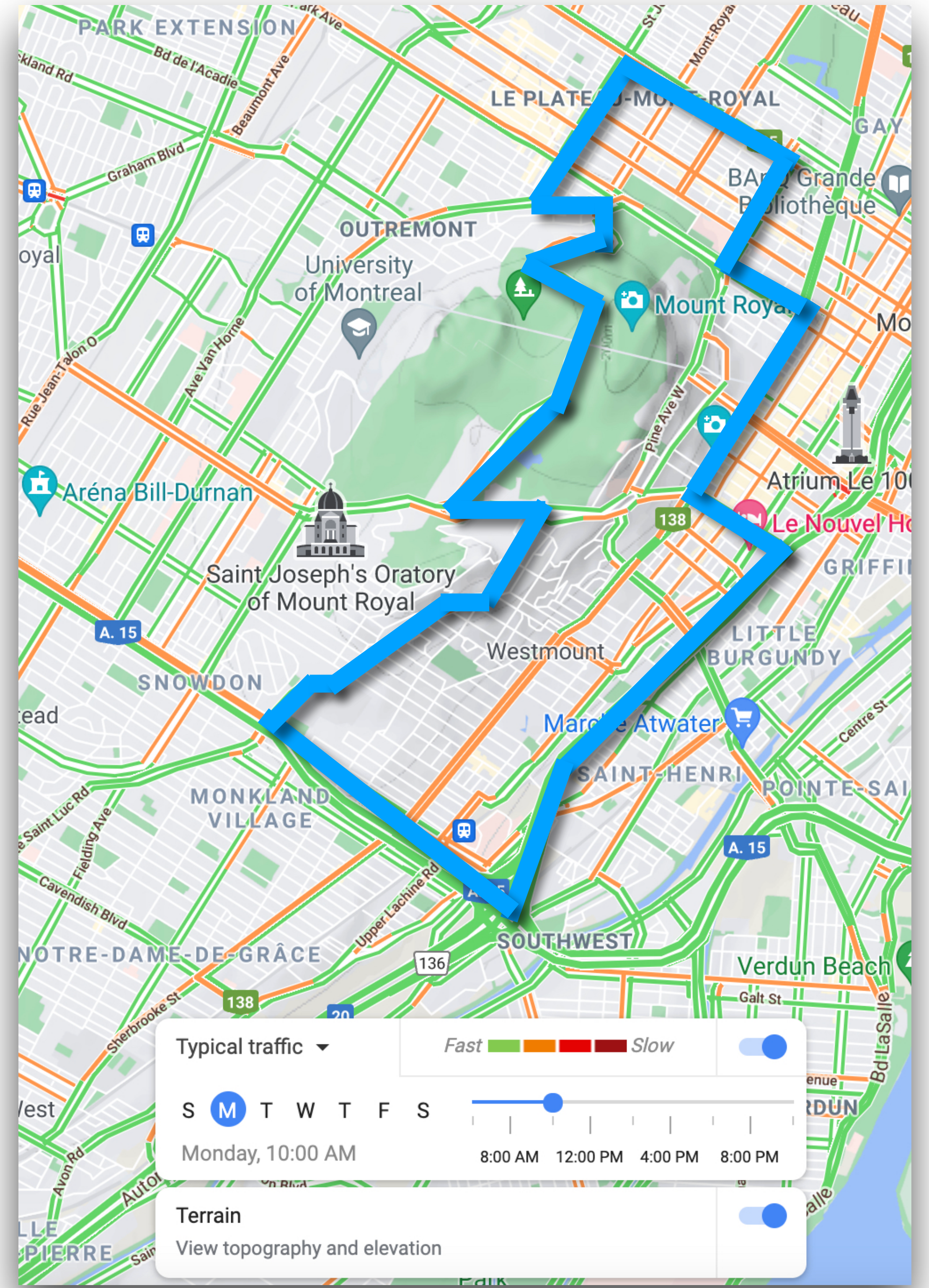
TSPs in Montréal

Every few hours, optimally **route** the salesperson through a set of locations

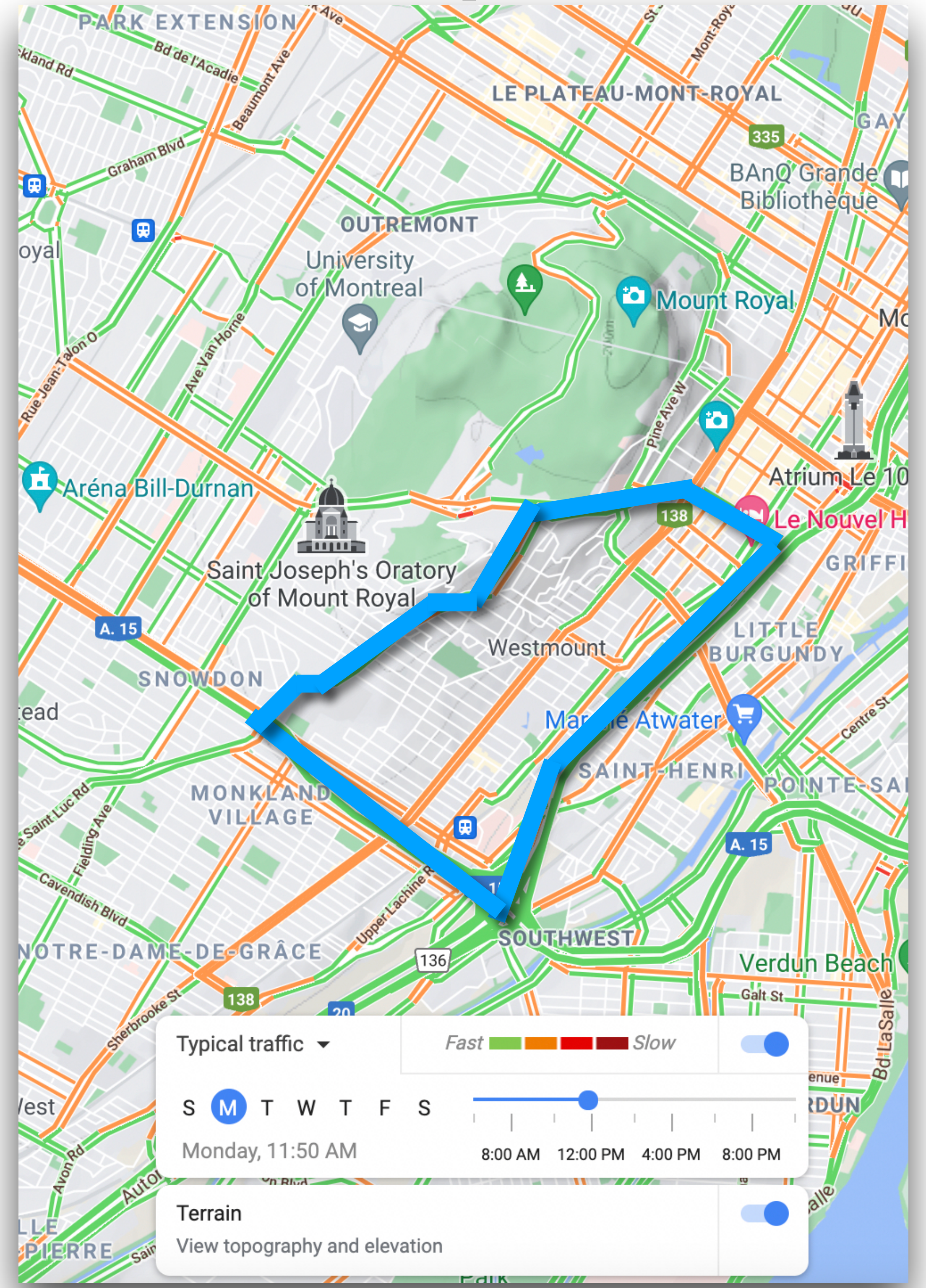
8 am



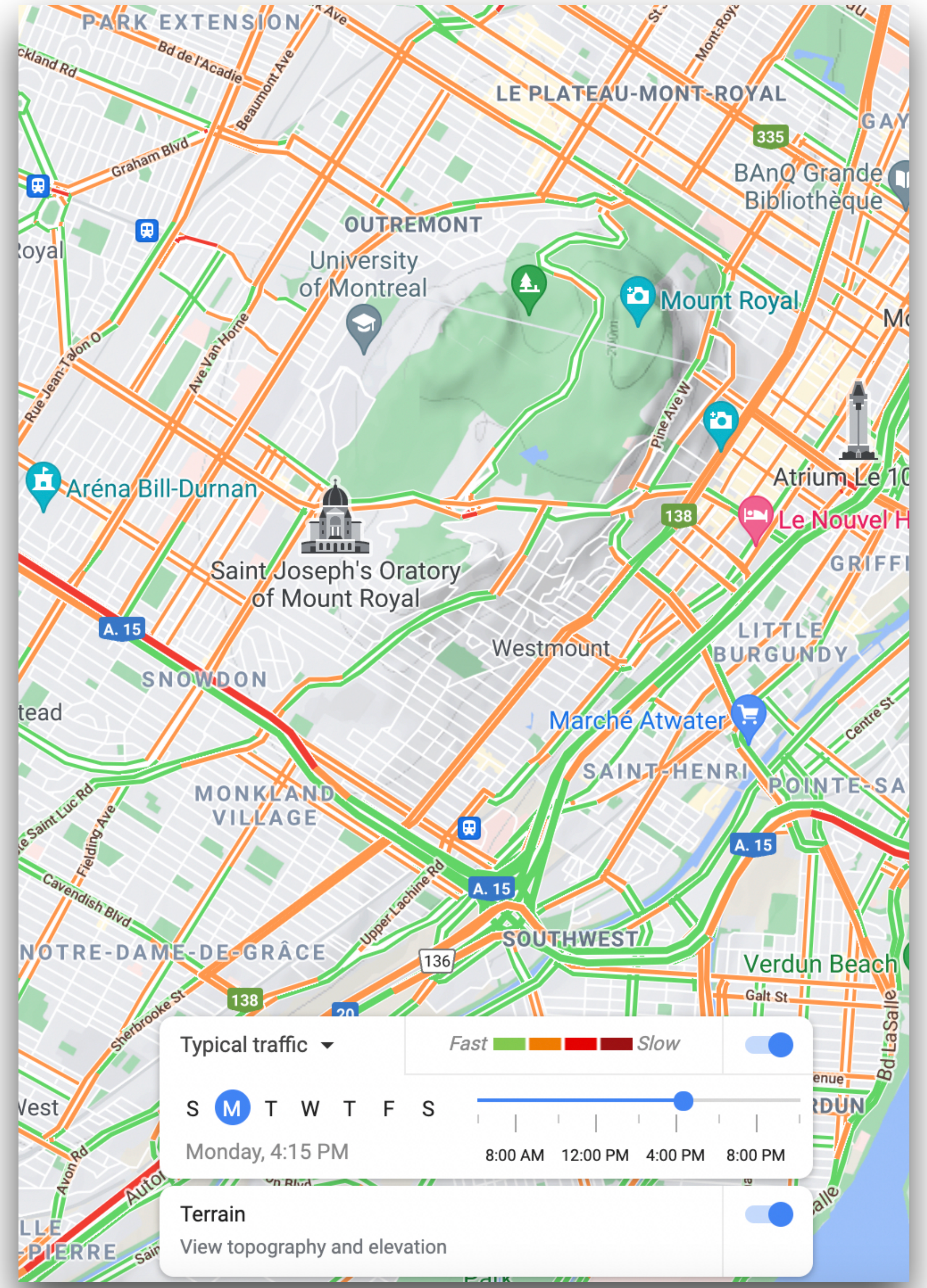
10 am



12 pm



4 pm

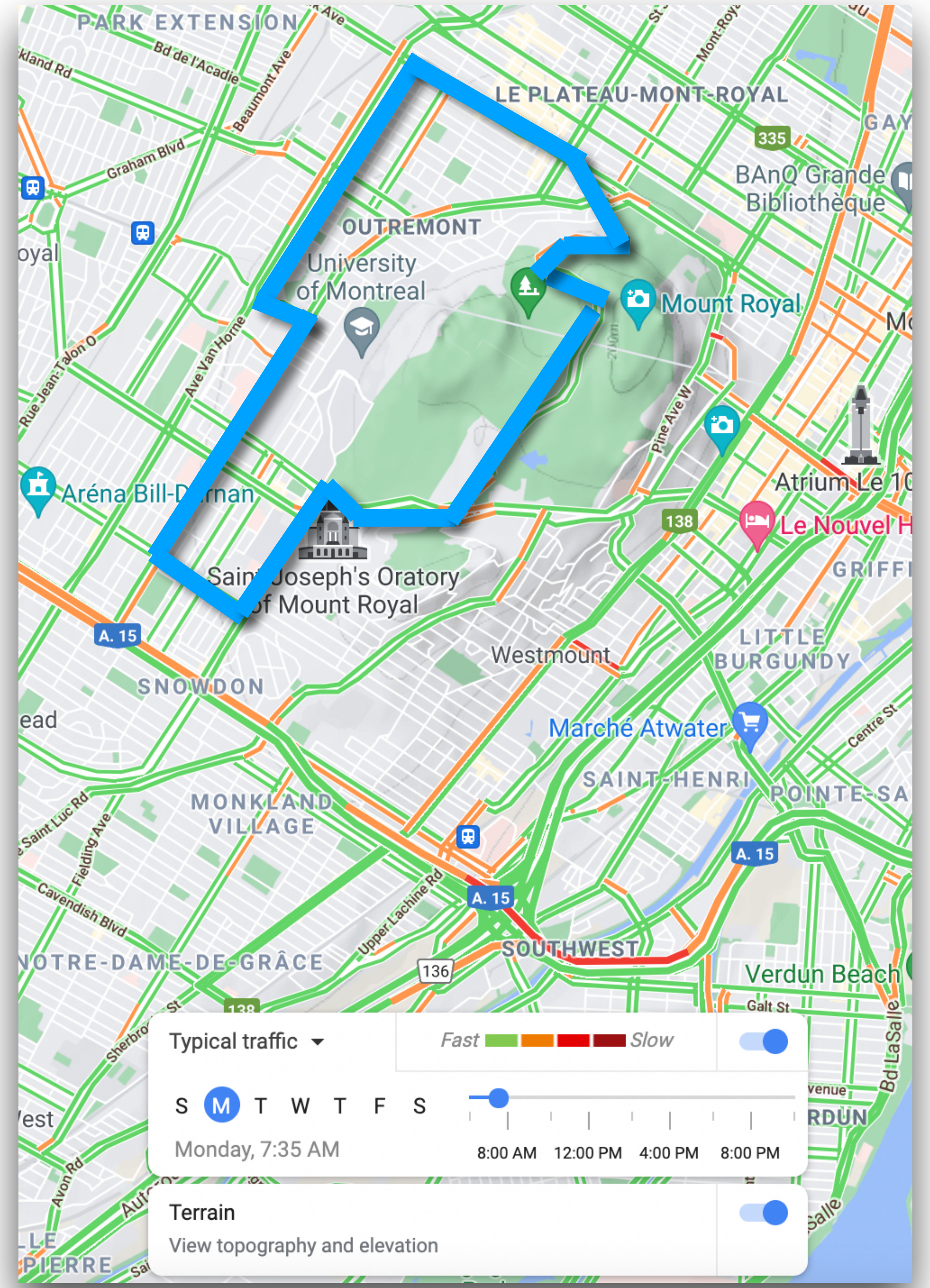


Google Maps, Montréal, Quebéc, Canada

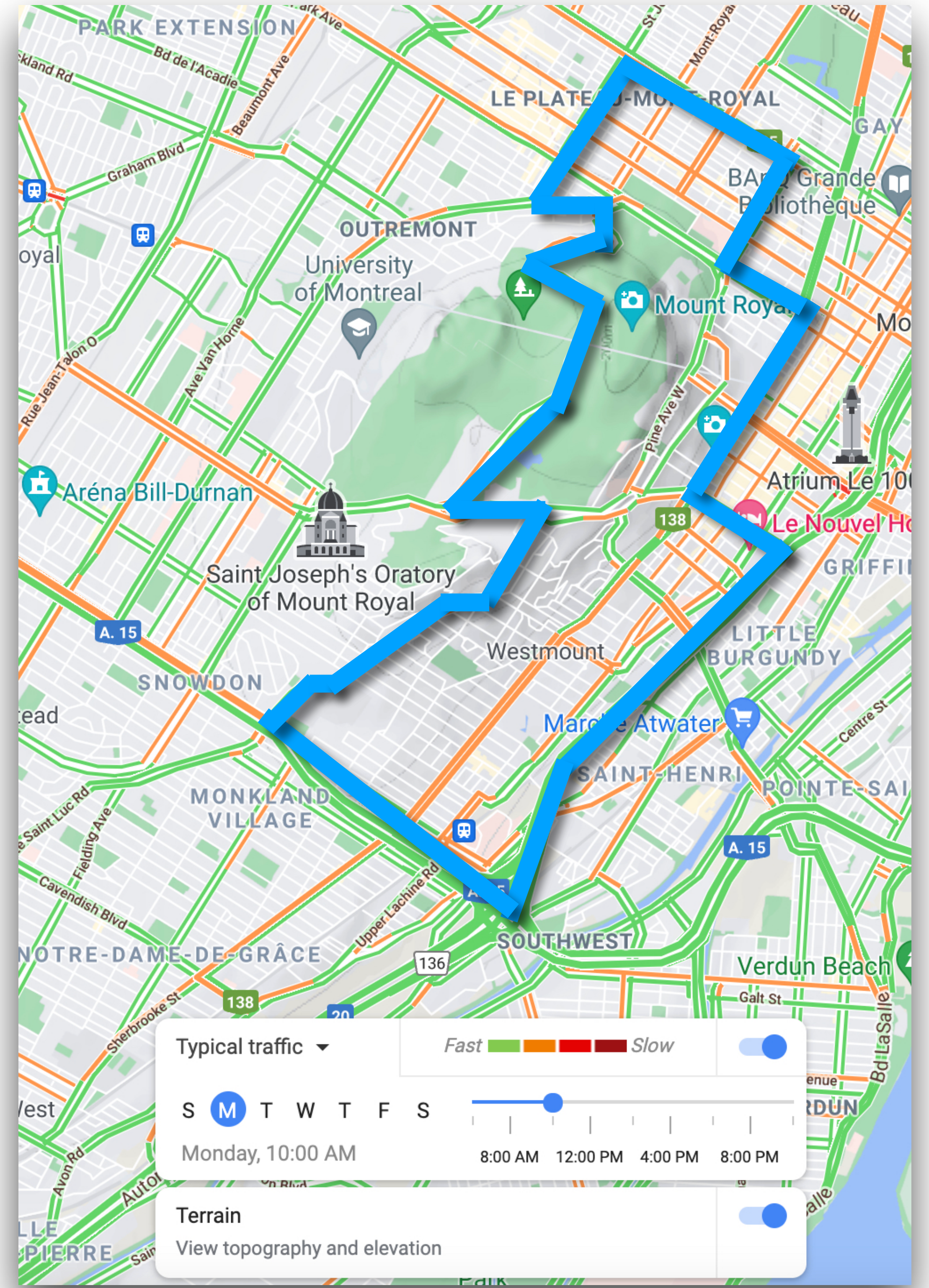
TSPs in Montréal

Every few hours, optimally **route** the salesperson through a set of locations

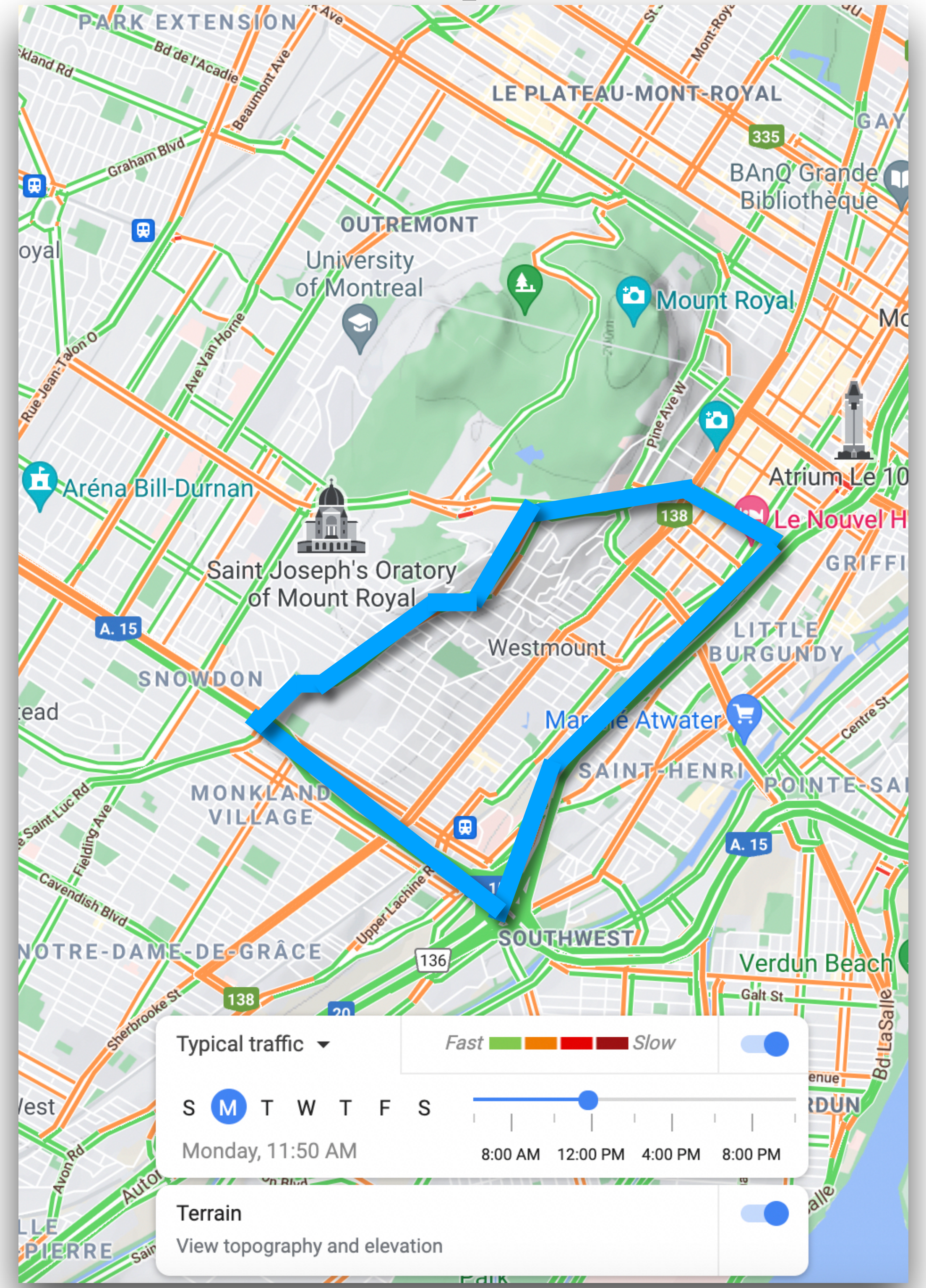
8 am



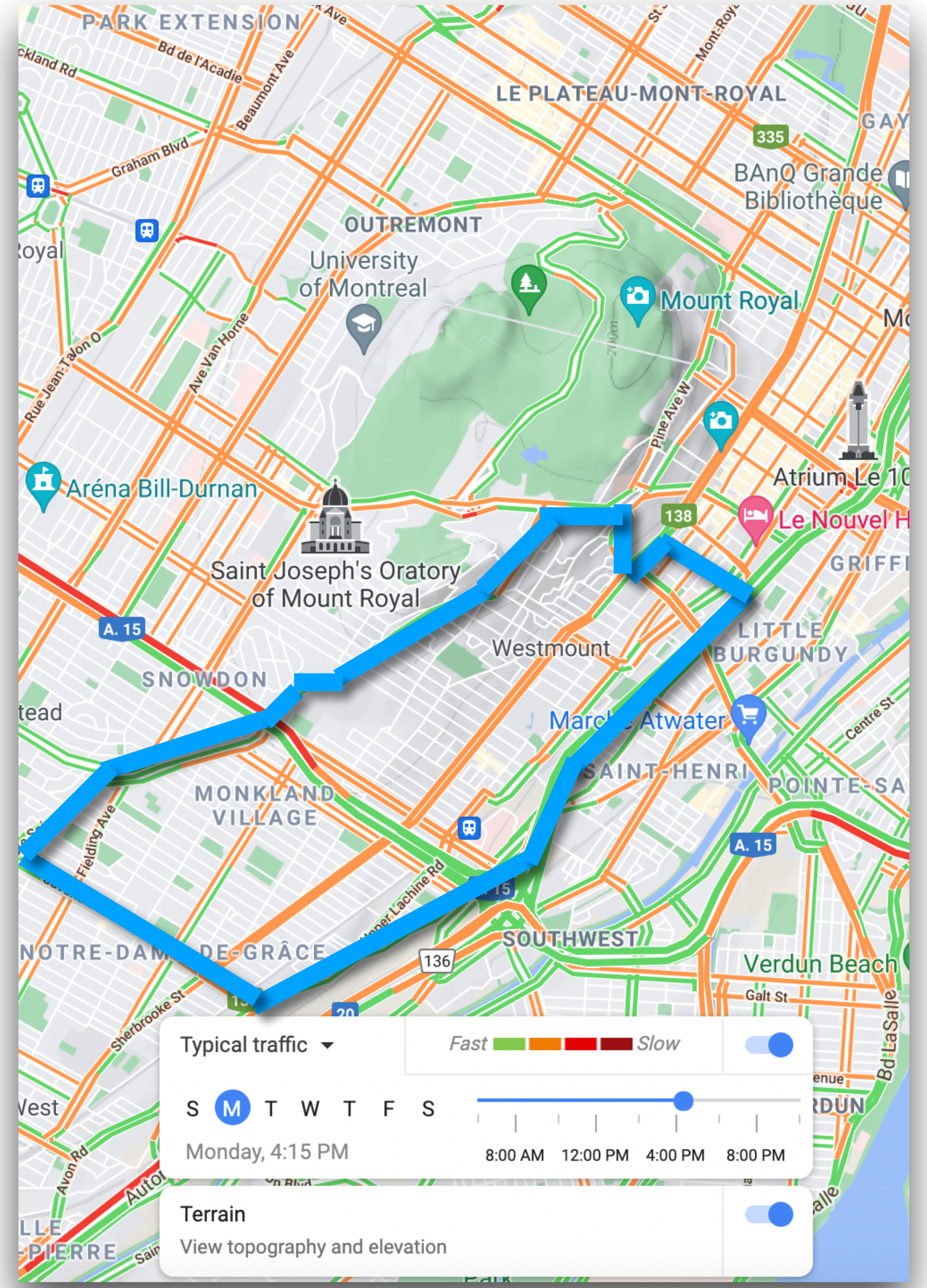
10 am



12 pm



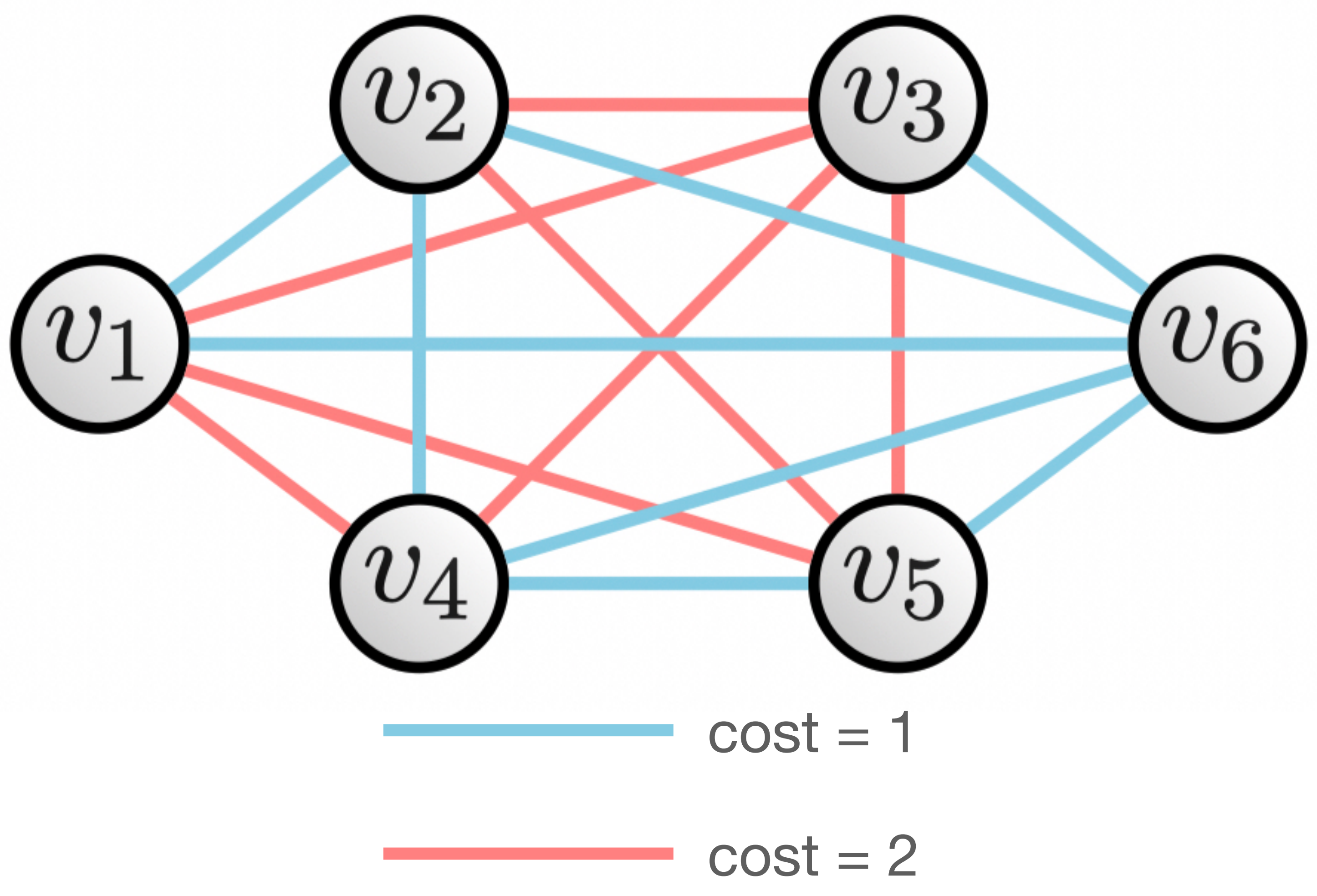
4 pm



Google Maps, Montréal, Quebéc, Canada

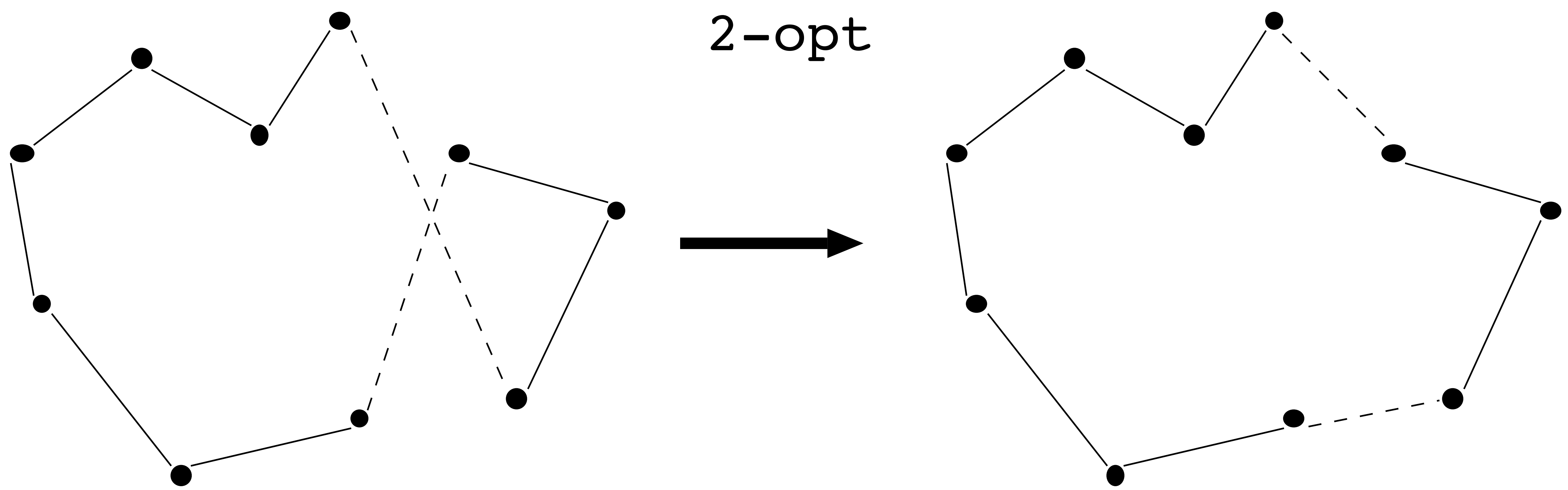
Nearest Neighbour heuristic for the TSP:

- always choose at the current city the closest unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the ith step choose city $\pi(i+1)$ to be the city j that minimises $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$



Iterative Improvement for the TSP

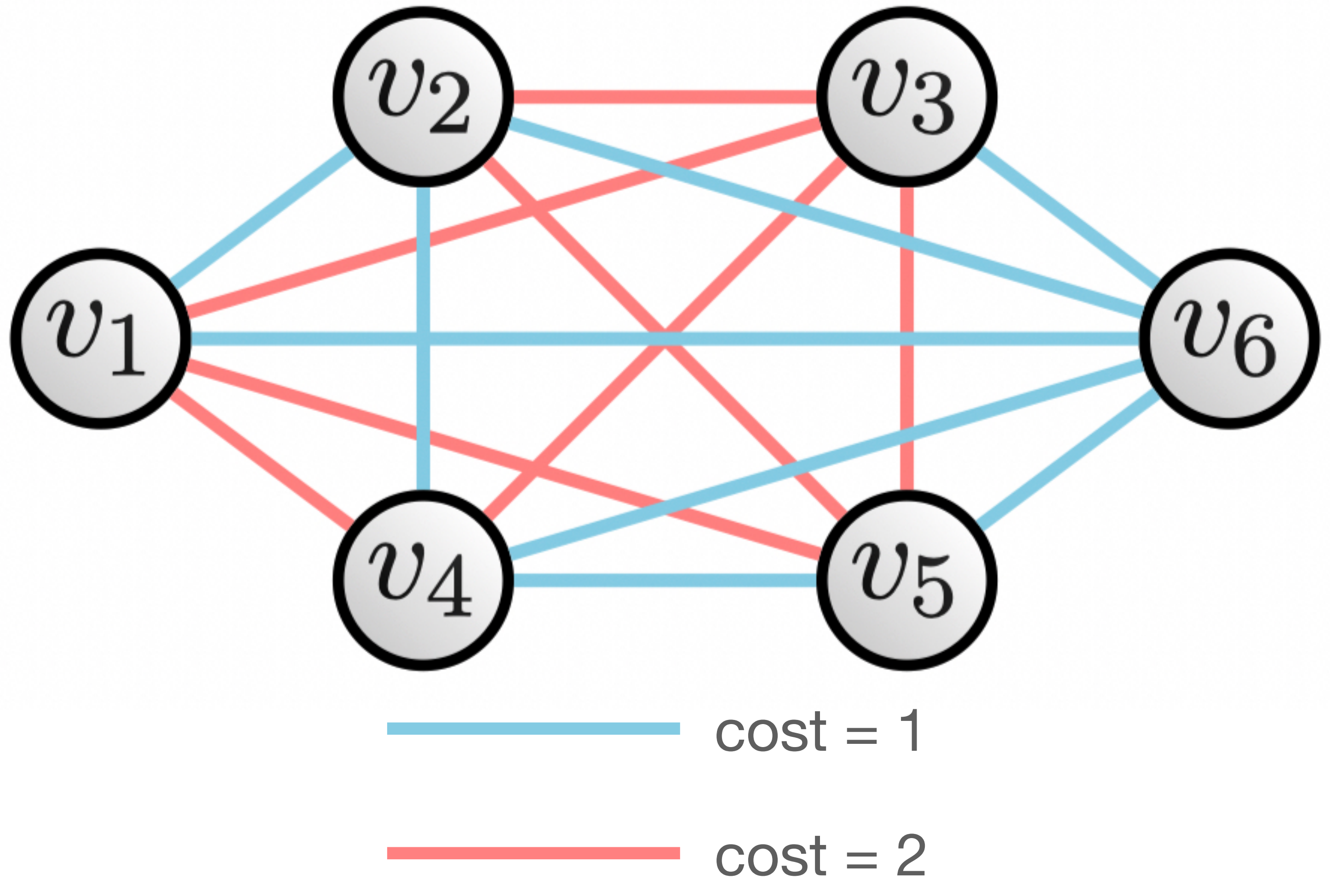
- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



- neighbourhood size $\mathcal{O}(n^k)$

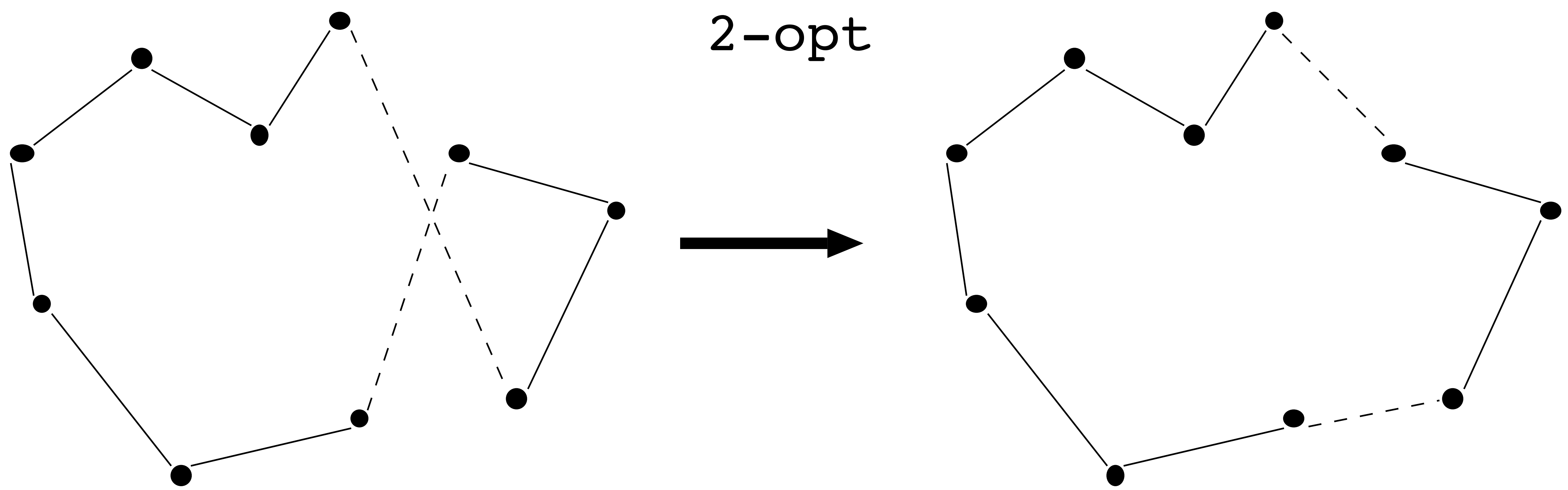
Nearest Neighbour heuristic for the TSP:

- always choose at the current city the closest unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the ith step choose city $\pi(i+1)$ to be the city j that minimises $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$



Iterative Improvement for the TSP

- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



Problems with local search?

- neighbourhood size $\mathcal{O}(n^k)$

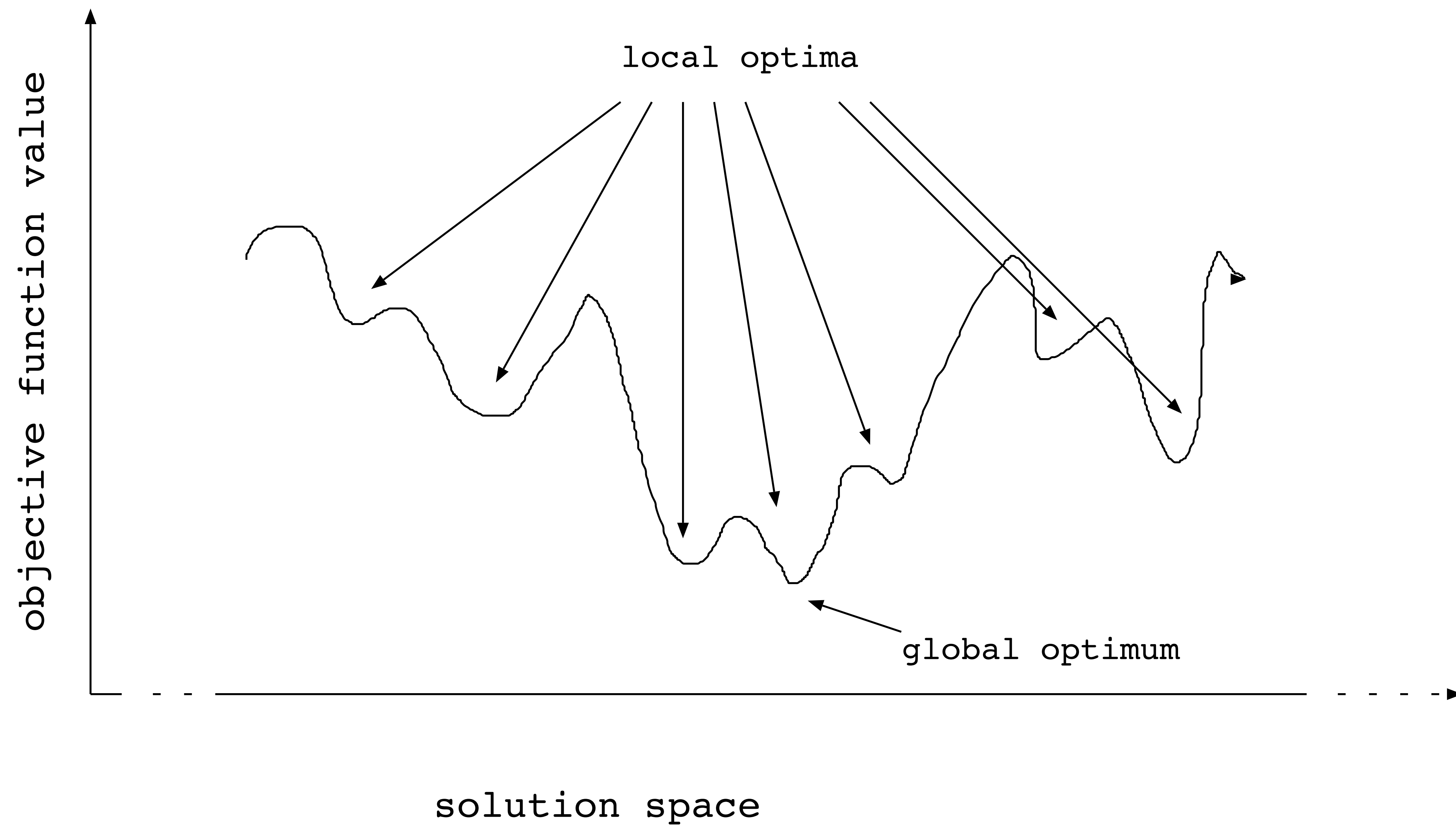
Stochastic Local Search:

- randomise initialisation step
  - random initial solutions
  - randomised construction heuristics

- randomise search steps
  such that suboptimal/worsening steps are allowed
  $\leadsto$ improved performance & robustness

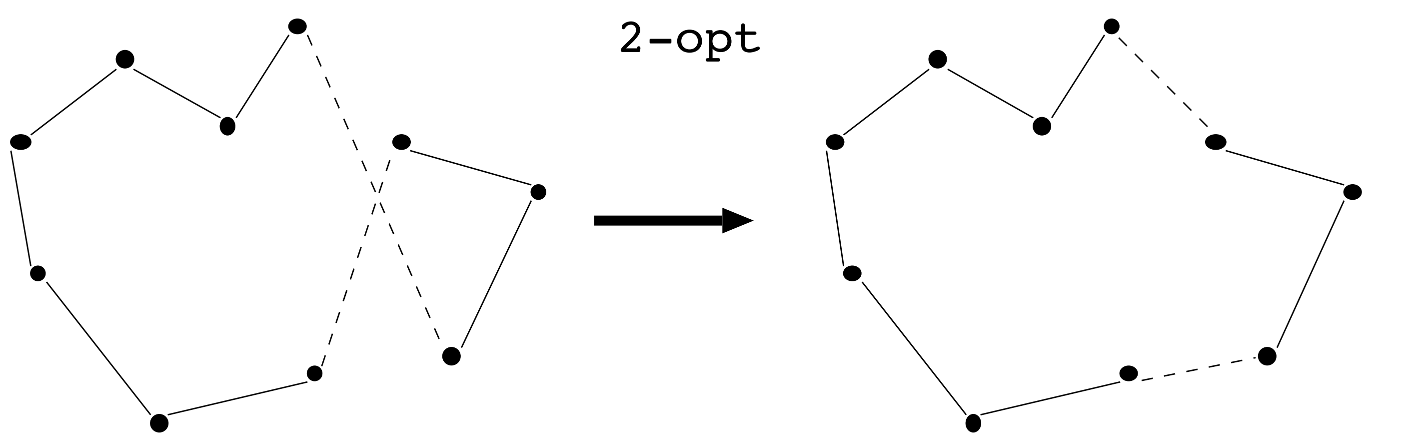
- typically, degree of randomisation controlled by noise parameter

- allows to invest arbitrary computation times



Iterative Improvement for the TSP

- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



0-1 Knapsack Problem

There is a budget b available for investment in projects during the coming year and n projects are under consideration, where a_j is the outlay for project j and c_j is its expected return. The goal is to choose a set of projects so that the budget is not exceeded and the expected return is maximized.

Definition of the variables.

$x_j = 1$ if project j is selected, and $x_j = 0$ otherwise.

Definition of the constraints.

The budget cannot be exceeded:

$$\sum_{j=1}^{n} a_j x_j \leq b.$$

The variables are 0–1:

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \ldots, n.$$

Definition of the objective function.

The expected return is maximized:

$$\max \sum_{j=1}^{n} c_j x_j.$$

Can you find a feasible solution greedily?

Knapsack

Greedy algorithm

Definition of the objective function.

The expected return is maximized:

$$\max \sum_{j=1}^{n} c_j x_j.$$

The budget cannot be exceeded:

$$\sum_{j=1}^{n} a_j x_j \leq b.$$

The variables are 0–1:

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \ldots, n.$$

Sort items in increasing order of $\dfrac{c_j}{a_j}$

While budget $< b$ and there is an item that fits:

Insert next best item from sorted list into knapsack; update budget

Knapsack

Greedy algorithm

$$\max \sum_{j=1}^{n} c_j x_j. \quad \text{s.t.} \sum_{j=1}^{n} a_j x_j \leq b. \quad x_j \in \{0,1\} \quad \text{for } j = 1,\ldots,n.$$

Sort items in increasing order of $\dfrac{c_j}{a_j}$

While budget $< b$ and there is an item that fits:

Insert next best item from sorted list into knapsack; update budget

$$b = 5$$

j	c_j	a_j	c_j/a_j
1	6	1	6
2	10	2	5
3	12	3	4

Knapsack

Greedy algorithm

$$\max \sum_{j=1}^{n} c_j x_j. \quad \text{s.t.} \sum_{j=1}^{n} a_j x_j \leq b. \quad x_j \in \{0,1\} \quad \text{for } j = 1, \ldots, n.$$

Sort items in increasing order of $\dfrac{c_j}{a_j}$

While budget $< b$ and there is an item that fits:

Insert next best item from sorted list into knapsack; update budget

$$b = 5$$

j	c_j	a_j	c_j/a_j	$c_j/a_j^{0.5}$
1	6	1	6	6
2	10	2	5	7.07
3	12	3	4	6.9

Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \le b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 Select Node

2 Solve LP Relaxation

3 Prune?

4 Add Cuts

5 Run Heuristics

6 Branch

N_0

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \cancel{\{0,1\}}^n$$

$$[0,1]^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Solve LP Relaxation
→ Lower Bound on OPT

Branch & Bound for Integer Optimization

LP-based

Land & Doig, 1960

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \cancel{\{0,1\}}^n$$

$[0,1]^n$

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Solve LP Relaxation

→ Lower Bound on OPT

worse than best solution?

Prune!

Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \leq b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Add Cuts:
Tightening Constraints

Branch & Bound for Integer Optimization

LP-based

$$\min_{x} c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

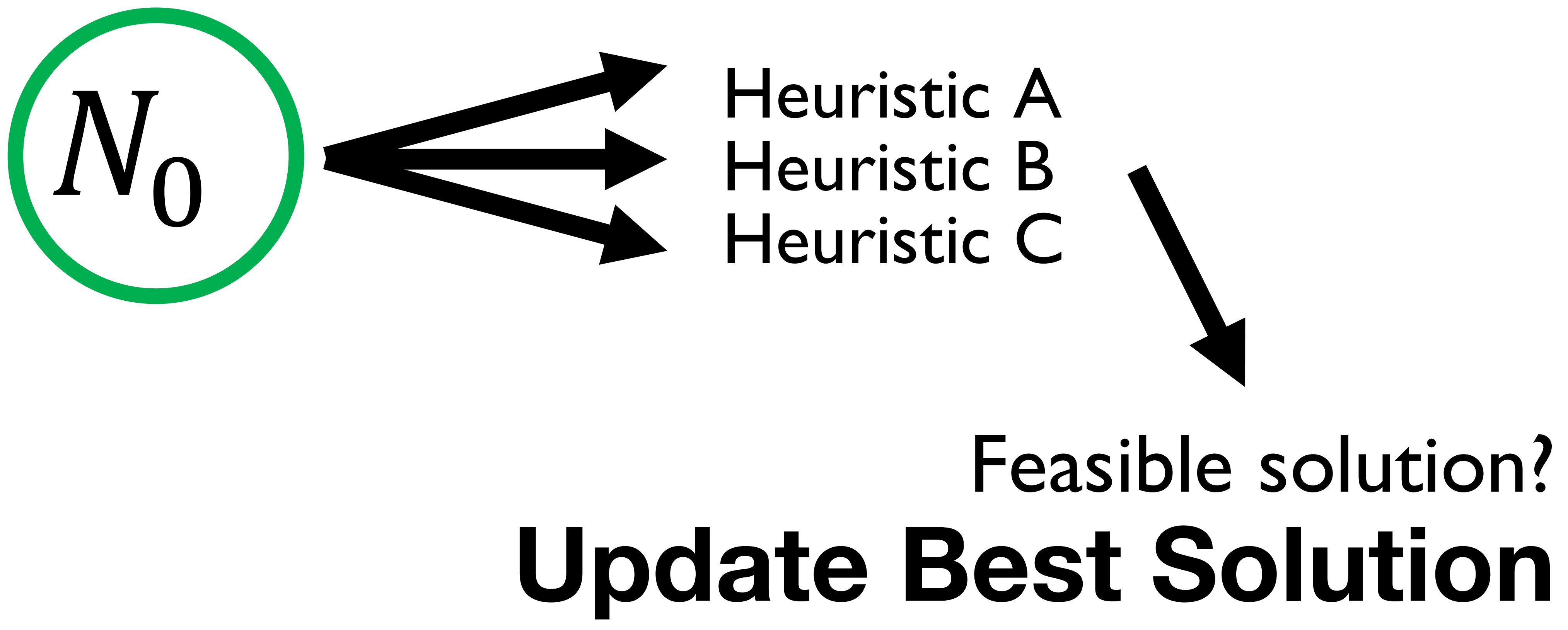
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \leq b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

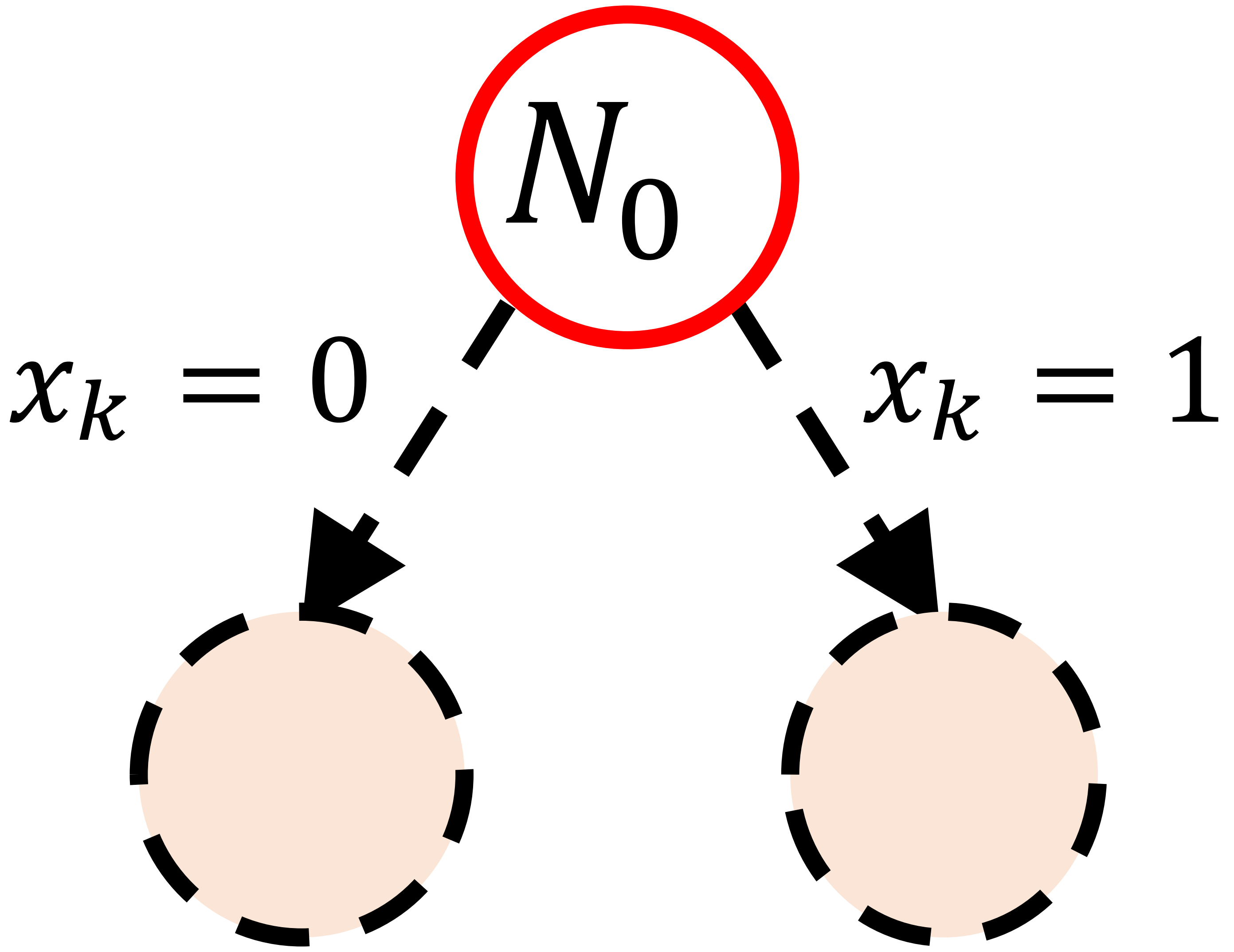
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



x_1?
x_2?
...
x_n?

Branch & Bound for Integer Optimization

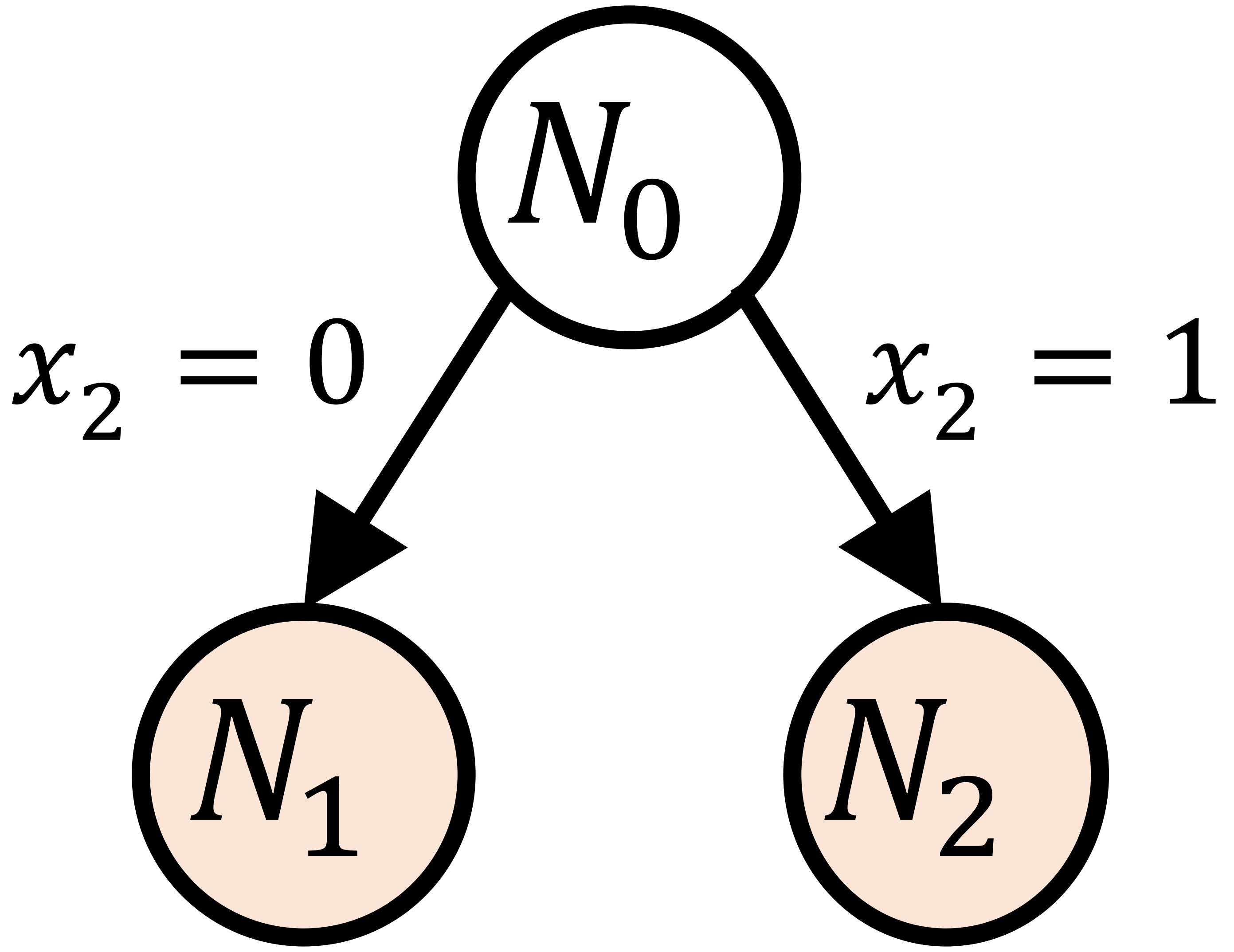
LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**
2 **Solve LP Relaxation**
3 **Prune?**
4 **Add Cuts**
5 **Run Heuristics**
6 **Branch**



x_1?
$\boxed{x_2}$
...
x_n?

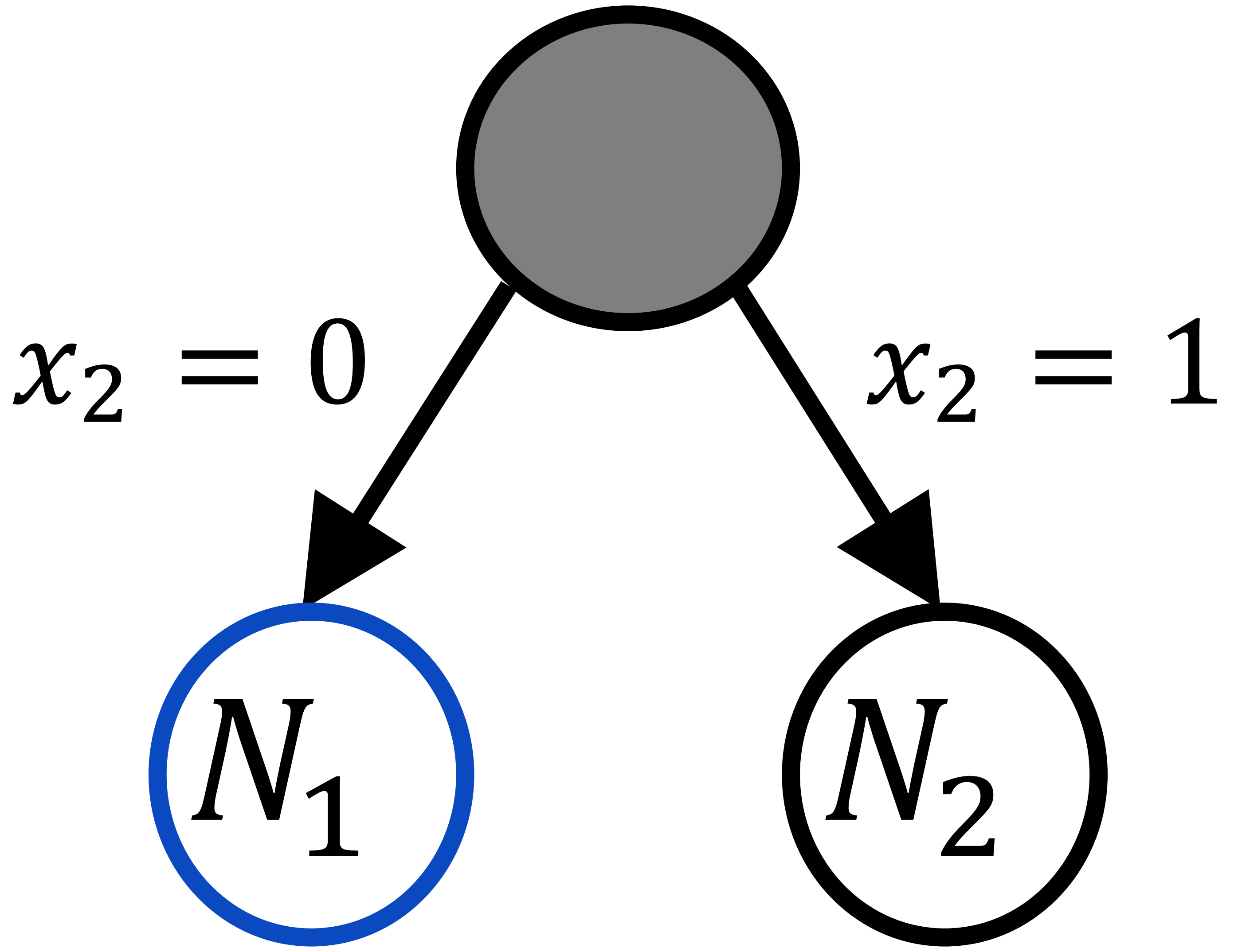
Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \le b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 Select Node
2 Solve LP Relaxation
3 Prune?
4 Add Cuts
5 Run Heuristics
6 Branch



Branch & Bound for Integer Optimization

Land & Doig, 1960

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Repeat:

1 **Select Node**

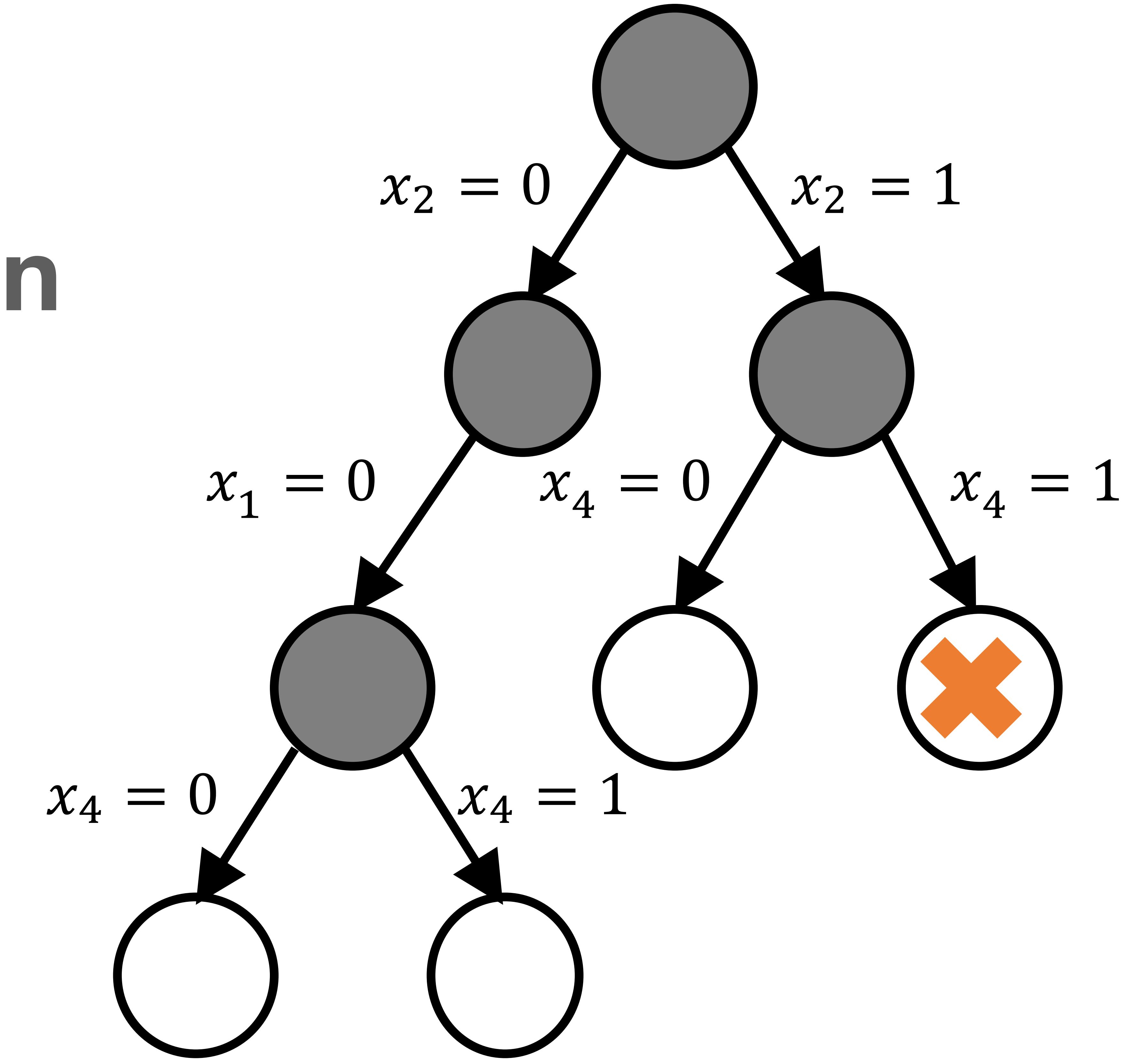
2 **Solve LP Relaxation**

3 **Prune?**

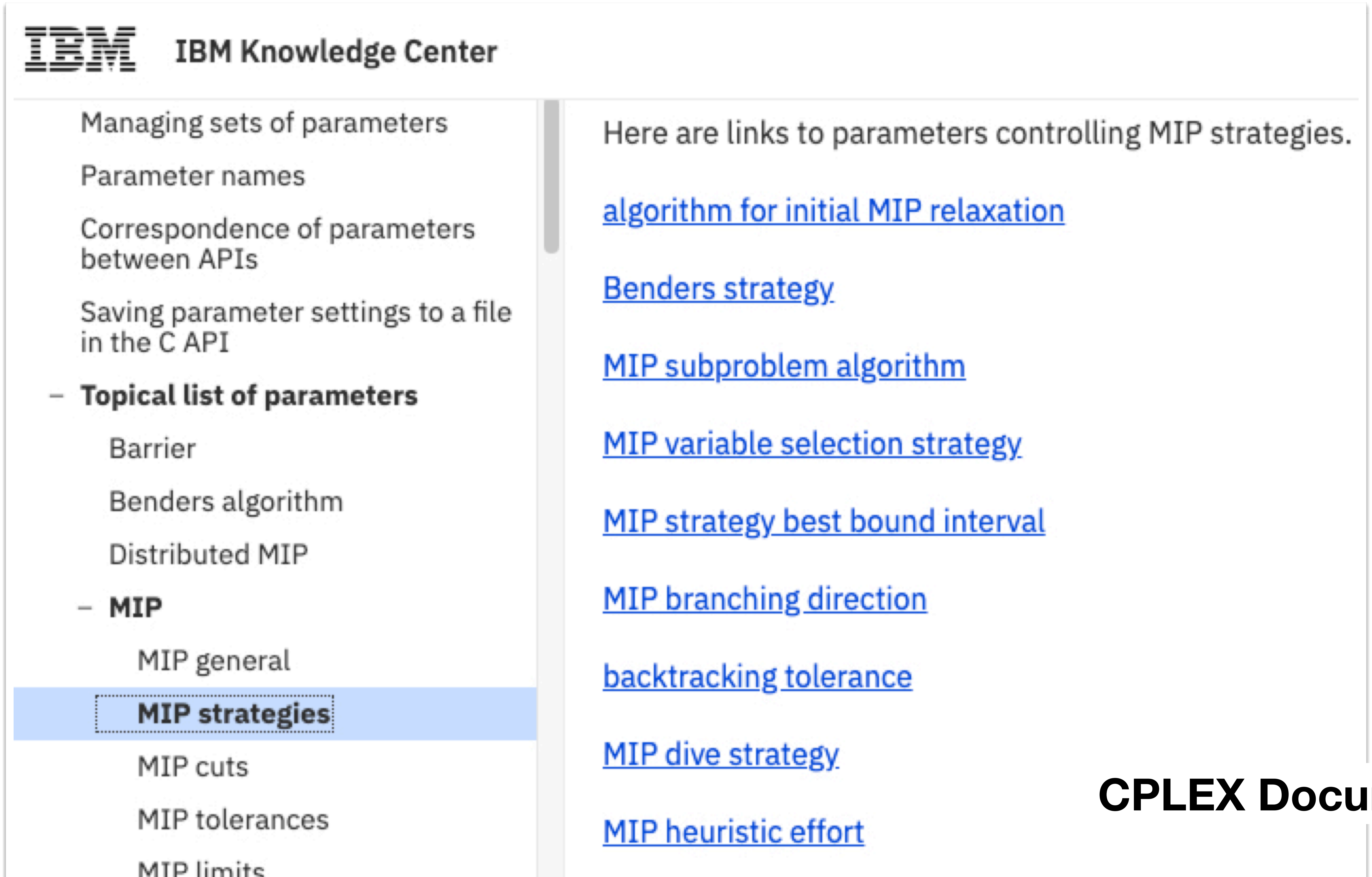
4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



Towards Tailored Algorithms



CPLEX Documentation

Towards Tailored Algorithms

MIP variable selection strategy

Value	Symbol	Meaning
-1	CPX_VARSEL_MININFEAS	Branch on variable with minimum infeasibility
0	CPX_VARSEL_DEFAULT	Automatic: let CPLEX choose variable to branch on; **default**
1	CPX_VARSEL_MAXINFEAS	Branch on variable with maximum infeasibility
2	CPX_VARSEL_PSEUDO	Branch based on pseudo costs
3	CPX_VARSEL_STRONG	Strong branching
4	CPX_VARSEL_PSEUDOREDUCED	Branch based on pseudo reduced costs

MIP heuristic frequency

Value	Meaning
-1	None
0	Automatic: let CPLEX choose; **default**
Any positive integer	Apply the periodic heuristic at this frequency

CPLEX Documentation

Hoos, Holger H. "Automated algorithm configuration and parameter tuning." Autonomous search. Springer, Berlin, Heidelberg, 2011. 37-71.

- an algorithm A with parameters $p_1, \ldots, p_k$ that affect its behaviour,
- a space C of parameter settings (configurations), where $c \in C$ specifies values for $p_1, \ldots, p_k$,
- a set of problem instances I,
- a performance metric m that measures the performance of A on instance set I for a given configuration c,

find a configuration $c^* \in C$ such that running algorithm A on instance set I maximizes metric m

Hoos, Holger H. "Automated algorithm configuration and parameter tuning." Autonomous search. Springer, Berlin, Heidelberg, 2011. 37-71.

Greedy 0-1 knapsack

Exponent p_1 such that items are sorted w.r.t. $c_j/a_j^{p_1}$

- an algorithm A with parameters $p_1, \ldots, p_k$ that affect its behaviour,

$p_1 \in (0,1]$

- a space C of parameter settings (configurations), where $c \in C$ specifies values for $p_1, \ldots, p_k$,

- a set of problem instances I,

$$\sum_{i \in I} \sum_{j=1}^{n} c_j x_j$$

- a performance metric m that measures the performance of A on instance set I for a given configuration c,

find a configuration $c^* \in C$ such that running algorithm A on instance set I maximizes metric m

Issues to consider

- Generalization
- Time-outs!
- **Optimization!**

Racing Procedures

123rf.com

- Assumptions: small, finite configuration space C
- Basic idea:
  - sample instance,
  - test remaining configs.,
  - eliminate really bad ones relative to current best config.,
  - repeat.

procedure *F-Race*
input *target algorithm A, set of configurations C, set of problem instances I,*
performance metric m;
parameters *integer* ni_{min};
output *set of configurations* C^*;

$C^* := C$; $ni := 0$;
repeat
randomly choose instance i from set I;
run all configurations of A in C^* on i;
$ni := ni + 1$;
if $ni \geq ni_{min}$ **then**
perform rank-based Friedman test on results for configurations in C^* on all instances
in I evaluated so far;
if test indicates significant performance differences **then**
c^* := best configuration in C^* (according to m over instances evaluated so far);
for all $c \in C^* \setminus \{c^*\}$ **do**
perform pairwise Friedman post hoc test on c and c^*;
if test indicates significant performance differences **then**
eliminate c from C^*;
end if;
end for;
end if;
end if;
until termination condition met;
return C^*;
end *F-Race*

123rf.com

Birattari, Mauro, et al. "A Racing Algorithm for Configuring Metaheuristics." GECCO. Vol. 2. No. 2002.

```
procedure F-Race
    input target algorithm A, set of configurations C, set of problem instances I,
          performance metric m;
    parameters integer ni_min;
    output set of configurations C*;

    C* := C;  ni := 0;
    repeat
        randomly choose instance i from set I;
        run all configurations of A in C* on i;
        ni := ni + 1;
        if ni ≥ ni_min then
            perform rank-based Friedman test on results for configurations in C* on all instances
                in I evaluated so far;
            if test indicates significant performance differences then
                c* := best configuration in C* (according to m over instances evaluated so far);
                for all c ∈ C* \ {c*} do
                    perform pairwise Friedman post hoc test on c and c*;
                    if test indicates significant performance differences then
                        eliminate c from C*;
                    end if;
                end for;
            end if;
        end if;
    until termination condition met;
    return C*;
end F-Race
```

123rf.com

Birattari, Mauro, et al. "A Racing Algorithm for Configuring Metaheuristics." GECCO. Vol. 2. No. 2002.

procedure *F-Race*
input *target algorithm A, set of configurations C, set of problem instances I, performance metric m*;
parameters *integer* ni_{min};
output *set of configurations* C^*;

$C^* := C;\ ni := 0;$
repeat
randomly choose instance i from set I;
run all configurations of A in C^* on i;
$ni := ni + 1;$
if $ni \geq ni_{min}$ **then**
perform rank-based Friedman test on results for configurations in C^* on all instances in I evaluated so far;
if test indicates significant performance differences **then**
c^* := best configuration in C^* (according to m over instances evaluated so far);
for all $c \in C^* \setminus \{c^*\}$ **do**
perform pairwise Friedman post hoc test on c and c^*;
if test indicates significant performance differences **then**
eliminate c from C^*;
end if;
end for;
end if;
end if;
until termination condition met;
return C^*;
end *F-Race*

Are these results statistically "similar"?

	Config X	Config Y	Config Z
Instance 1	7	5	8
Instance 2	1	4	2
Instance 3	3	3	3
Instance 4	2	7	8

Sum of ranks	4	7	9

$$T = \frac{(n-1)\sum_{j=1}^{n}\left(R_j - \frac{k(n+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}}$$

Birattari, Mauro, et al. "A Racing Algorithm for Configuring Metaheuristics." GECCO. Vol. 2. No. 2002.

procedure *I/F-Race*

input *target algorithm A, set of configurations C, set of problem instances I, performance metric m*;

output *set of configurations* C^*;

initialise probabilistic model M;

$C' := \emptyset$; *// later, C' is the set of survivors from the previous F-Race*

repeat

based on model M, sample set of configurations $\widehat{C} \subseteq C$;

perform F-Race on configurations in $\widehat{C} \cup C'$ to obtain set of configurations C^*;

update probabilistic model M based on configurations in C^*;

$C' := C^*$;

until termination condition met;

return $c^* \in C^*$ with best performance (according to m) over all instances evaluated;

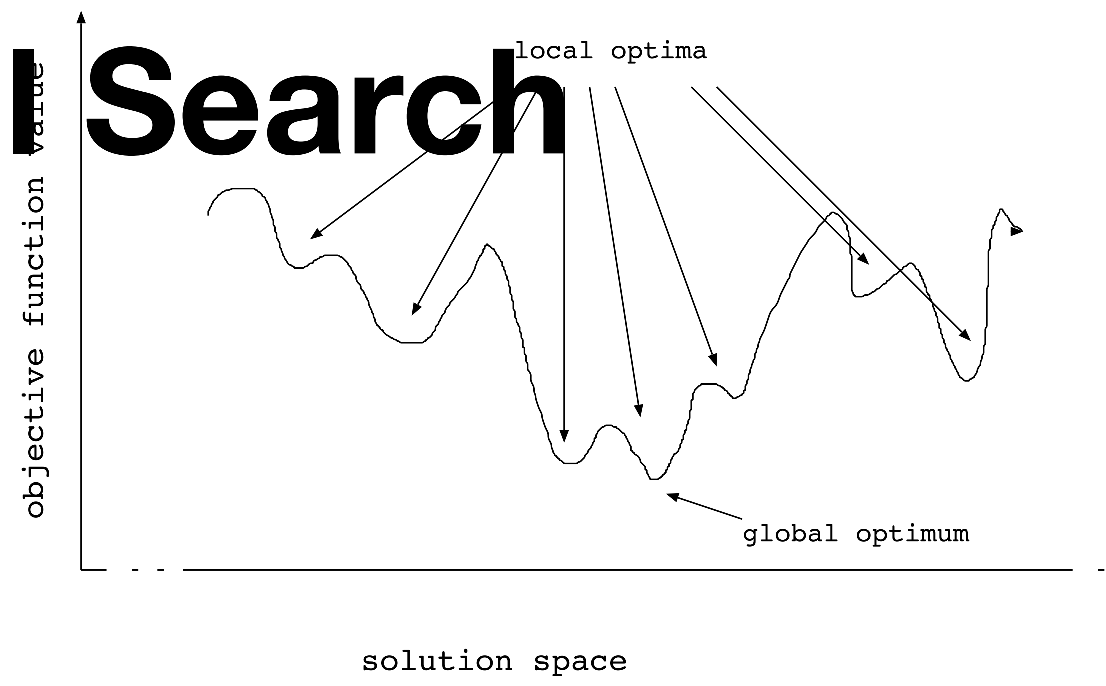
end *I/F-Race*

123rf.com

Balaprakash, Prasanna, Mauro Birattari, and Thomas Stützle. "Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement." International workshop on hybrid metaheuristics. Springer, Berlin, Heidelberg, 2007.

ParamILS

ILS: Iterated Local Search



procedure *ParamILS*
  input *target algorithm A, set of configurations C, set of problem instances I,*
    performance metric m;
  parameters *configuration* $c_0 \in C$, *integer r, integer s, probability pr*;
  output *configuration* c^*;

  $c^* := c_0$;
  for $i := 1$ **to** r **do**
    draw c from C uniformly at random;
    assess c against c^* based on performance of A on instances from I according to metric m;
    if c found to perform better than c^* **then**
      $c^* := c$;
    end if;
  end for;

  $c := c^*$;
  perform subsidiary local search on c;
  while termination condition not met **do**
    $c' := c$;
    perform s random perturbation steps on c'
    perform subsidiary local search on c';
    assess c' against c based on performance of A on instances from I according to metric m;
    if c' found to perform better than c **then**   *// acceptance criterion*
      update overall incumbent c^*;
      $c := c'$;
    end if;
    with probability *pr* **do**
      draw c from C uniformly at random;
    end with probability;
  end while;
  return c^*;
end *ParamILS*

Initial sampling phase

Random perturbation + local search

Evaluation

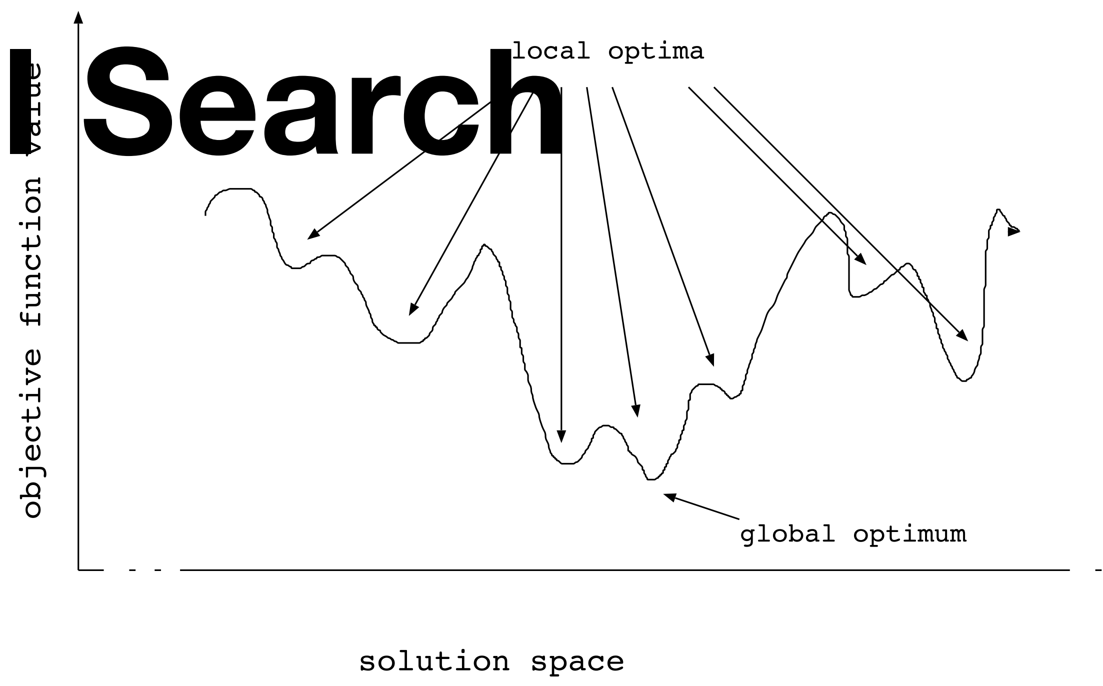
Update incumbent config.

Random restart!

Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.

ParamILS

ILS: Iterated Local Search



procedure *ParamILS*
  input *target algorithm A, set of configurations C, set of problem instances I,*
      performance metric m;
  parameters *configuration* $c_0 \in C$*, integer r, integer s, probability pr*;
  output *configuration* c^*;

  $c^* := c_0$;
  for $i := 1$ **to** r **do**
    draw c from C uniformly at random;
    assess c against c^* based on performance of A on instances from I according to metric m;
    if c found to perform better than c^* **then**
      $c^* := c$;
    end if;
  end for;

Initial sampling phase

  $c := c^*$;
  perform subsidiary local search on c;
  while termination condition not met **do**
    $c' := c$;
    perform s random perturbation steps on c'
    perform subsidiary local search on c';
    assess c' against c based on performance of A on instances from I according to metric m;
    if c' found to perform better than c **then**   *// acceptance criterion*
      update overall incumbent c^*;
      $c := c'$;
    end if;
    with probability *pr* **do**
      draw c from C uniformly at random;
    end with probability;
  end while;
  return c^*;
end *ParamILS*

Random perturbation + local search
Evaluation
Update incumbent config.

Random restart!

Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.

Sequential Model-Based Optimization

procedure *SMBO*

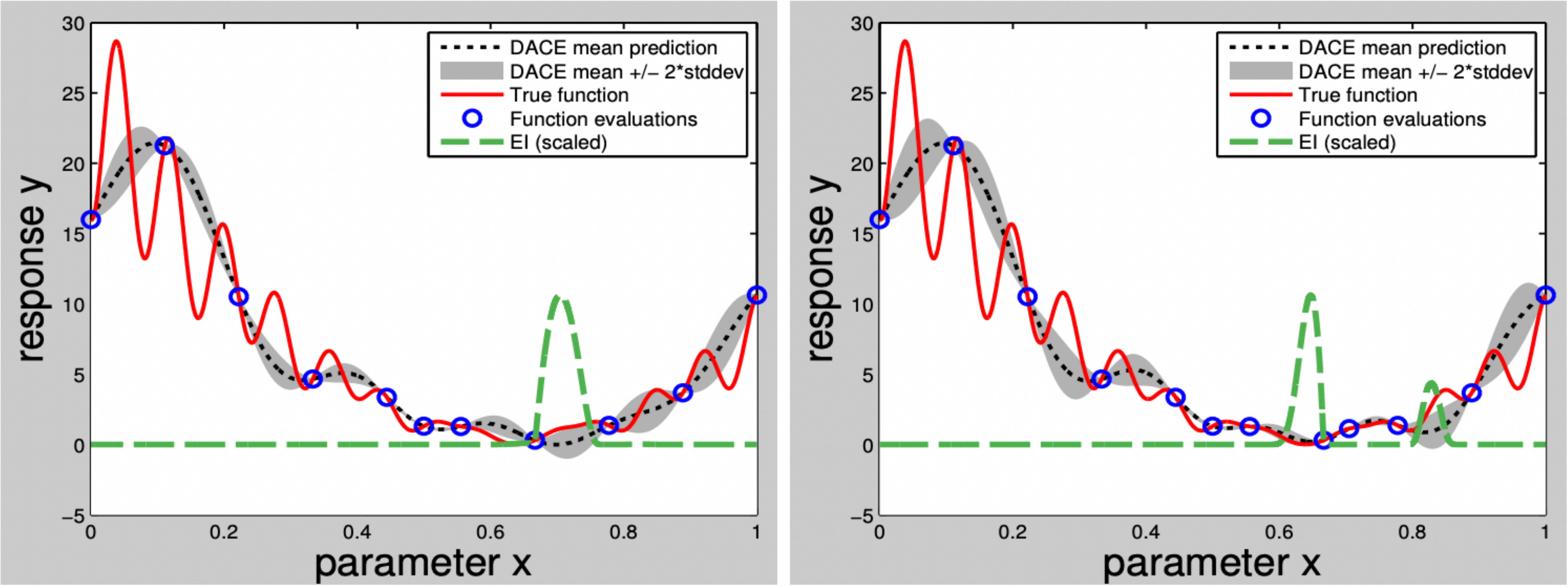
input *target algorithm A, set of configurations C, set of problem instances I,*
performance metric m;

output *configuration* c^*;

determine initial set of configurations $C_0 \subset C$;
for all $c \in C_0$, measure performance of A on I according to metric m;
build initial model M based on performance measurements for C_0;
determine incumbent $c^* \in C_0$ for which best performance was observed or predicted;
repeat
based on model M, determine set of configurations $C' \subseteq C$;
for all $c \in C'$, measure performance of A on I according to metric m;
update model M based on performance measurements for C';
update incumbent c^*;
until termination condition met;
return c^*;
end *SMBO*

Sequential Model-Based Optimization

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration." *International conference on learning and intelligent optimization*. Springer, Berlin, Heidelberg, 2011.



(a) SMBO, step 1

(b) SMBO, step 2

Fig. 1. Two steps of SMBO for the optimization of a 1D function. The true function is shown as a solid line, and the circles denote our observations. The dotted line denotes the mean prediction of a noise-free Gaussian process model (the "DACE" model), with the grey area denoting its uncertainty. Expected improvement (scaled for visualization) is shown as a dashed line.

Sequential Model-Based Optimization

procedure *SMBO*

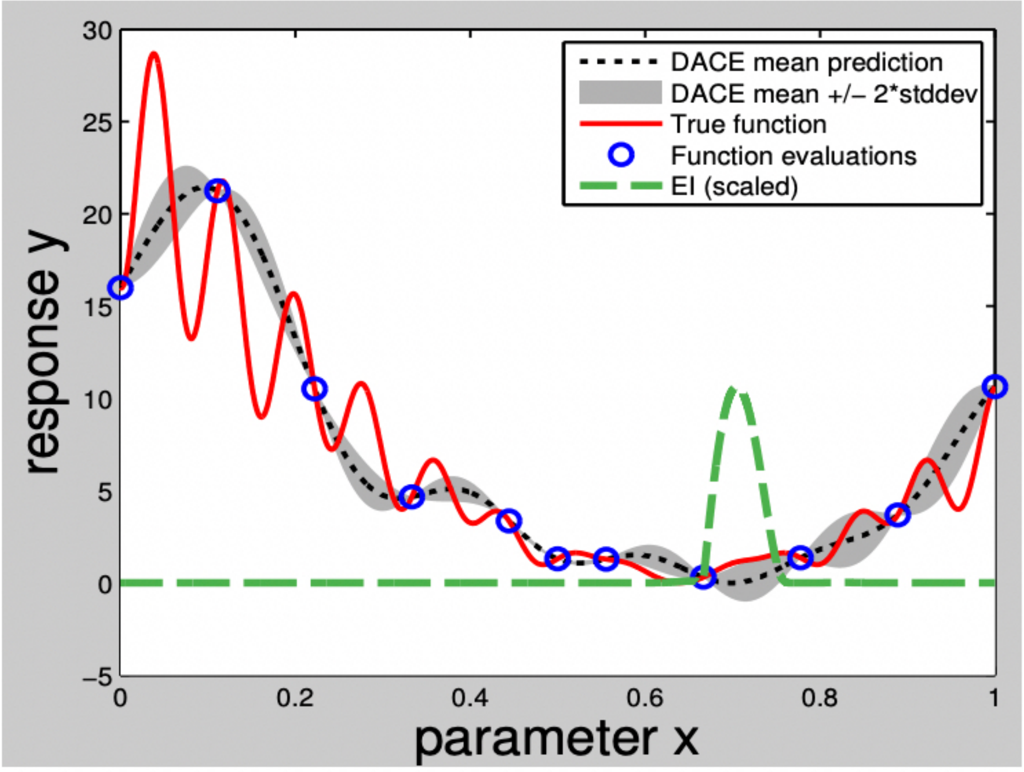
  input *target algorithm A, set of configurations C, set of problem instances I,*
    performance metric m;

  output *configuration* c^*;

  determine initial set of configurations $C_0 \subset C$;
  for all $c \in C_0$, measure performance of A on I according to metric m;
  build initial model M based on performance measurements for C_0;
  determine incumbent $c^* \in C_0$ for which best performance was observed or predicted;
  repeat
    based on model M, determine set of configurations $C' \subseteq C$;
    for all $c \in C'$, measure performance of A on I according to metric m;
    update model M based on performance measurements for C';
    update incumbent c^*;
  until termination condition met;
  return c^*;

end *SMBO*

$$EI(x) := E[\max\{f_{min} - \hat{F}(x), 0\}]$$



Hoos, Holger H. "Automated algorithm configuration and parameter tuning." Autonomous search. Springer, Berlin, Heidelberg, 2011. 37-71.

Other flavours of algo. config.

“

- **per-instance algorithm selection** methods choose one of several target algorithms to be applied to a given problem instance based on properties of that instance determined just before attempting to solve it

- **Reactive search procedures**, on-line algorithm control methods and adaptive operator selection techniques switch between different algorithms, heuristic mechanisms and parameter configurations while running on a given problem instance

- **dynamic algorithm portfolio** approaches repeatedly adjust the allocation of CPU shares between algorithms that are running concurrently on a given problem instance

”