ECE 602 – Section 9
Network Optimization (Part 2)

- **Objective:** maximize the divergence out of s over all capacity-feasible flow vectors x having zero divergence for all $i \in \mathcal{N} \backslash \{s, t\}$.
- Recall that "capacity-feasible" means

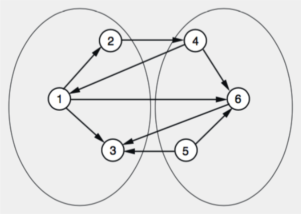
$$l_{ij} \leq x_{ij} \leq u_{ij}, \quad \forall (i, j) \in \mathcal{A}$$

- **The key idea:** A feasible flow x can be improved if we can find a path from s to t that is *unblocked* with respect to x.
- Note: pushing a positive increment of flow along such a path results in larger divergence out of s (i.e., y_s), while maintaining flow feasibility.
- The reverse is correct too: if we *cannot* find an unblocked path from s to t, the current flow is maximal.

- A *cut* $\mathcal{Q}$ in a graph $(\mathcal{N}, \mathcal{A})$ is a partition of $\mathcal{N}$ into $\mathcal{S}$ and $\mathcal{T} := \mathcal{N} \setminus \mathcal{S}$, namely $\mathcal{Q} = [\mathcal{S}, \mathcal{T}]$. In general, $[\mathcal{S}, \mathcal{T}] \neq [\mathcal{T}, \mathcal{S}]$.
- Let $\mathcal{Q}^+$ and $\mathcal{Q}^-$ be the *sets of forward and backward arcs of the cut* $\mathcal{Q}$, respectively, defined as

$$\mathcal{Q}^+ = \{(i,j) \in \mathcal{A} \mid i \in \mathcal{S}, j \in \mathcal{T}\}$$
$$\mathcal{Q}^- = \{(i,j) \in \mathcal{A} \mid i \in \mathcal{T}, j \in \mathcal{S}\}$$

- Q is called *non-empty* if $\mathcal{Q}^+ \cup \mathcal{Q}^- \neq \emptyset$ (otherwise, it is called *empty*).



$$Q = [\mathcal{S}, \mathcal{N} - \mathcal{S}],$$

where $\mathcal{S} = \{1, 2, 3\}$. We have

$$Q^+ = \{(2,4), (1,6)\},$$

$$Q^- = \{(4,1), (6,3), (5,3)\}.$$

- Given x, the *flux* across a nonempty cut $\mathcal{Q}$ is defined as

$$F(\mathcal{Q}) = \sum_{(i,j)\in\mathcal{Q}^+} x_{ij} - \sum_{(i,j)\in\mathcal{Q}^-} x_{ij} = \sum_{i\in\mathcal{S}} y_i$$

where y is the vector of divergences.

- Given lower and upper flow bounds l_{ij} and u_{ij}, the *capacity* of a nonempty cut $\mathcal{Q}$ is

$$C(\mathcal{Q}) = \sum_{(i,j)\in\mathcal{Q}^+} u_{ij} - \sum_{(i,j)\in\mathcal{Q}^-} l_{ij}$$

- For any capacity-feasible x, $F(\mathcal{Q}) \leq C(\mathcal{Q})$. If $F(\mathcal{Q}) = C(\mathcal{Q})$, then $\mathcal{Q}$ is said to be a *saturated cut with respect to* x.
- The flow of each forward (backward) arc of such a cut must be at its upper (lower) bound.

Proposition

Let x be capacity-feasible, and let s and t be two nodes. Then, either (1) $\exists$ a simple path from s to t which is unblocked w.r.t. x, **or** (2) $\exists$ a saturated cut that separates s from t.

- Consider the max-flow problem, where we want to maximize y_s over all capacity-feasible x for which $y_i = 0$ if $i \in \mathcal{N} \setminus \{s, t\}$.
- Given any such x and any cut $\mathcal{Q}$ separating s from t, we have

$$y_s = F(\mathcal{Q}) \leq C(\mathcal{Q})$$

- Thus, if the max-flow problem is feasible, we have

$$\text{Maximum Flow} \leq C(\mathcal{Q})$$

- The max-flow/min-cut theorem asserts that equality is attained for some $\mathcal{Q}$.

The max flow/min-cut theorem

(a) If x^* is an optimal solution of the max-flow problem, then y_s correspon- ding to x^* is equal to the minimum cut capacity over all $\mathcal{Q}$ separating s from t.

(b) If $l_{ij} = 0, \forall (i, j)$, the max-flow problem has an optimal solution, and the maximal y_s is equal to the minimum cut capacity over all $\mathcal{Q}$ separating s from t.

- **The key idea:** given a feasible x and a path P from s to t that is unblocked w.r.t. x, we can increase x_{ij} over P^+ and decrease x_{ij} over P^-.
- The maximum increment of flow change is

$$\delta = \min\left\{\{u_{ij} - x_{ij} \mid (i,j) \in P^+\}, \{x_{ij} - l_{ij} \mid (i,j) \in P^-\}\right\}$$

- The resulting flow vector $\bar{x}$ is given by

$$\bar{x}_{ij} = \begin{cases} x_{ij} + \delta & \text{if } (i,j) \in P^+ \\ x_{ij} - \delta & \text{if } (i,j) \in P^- \\ x_{ij} & \text{otherwise} \end{cases}$$

- $\bar{x}$ is feasible, and it has y_s that is larger by δ than y_s related to x.
- The operation of replacing x by $\bar{x}$ is called a *flow augmentation along* P.

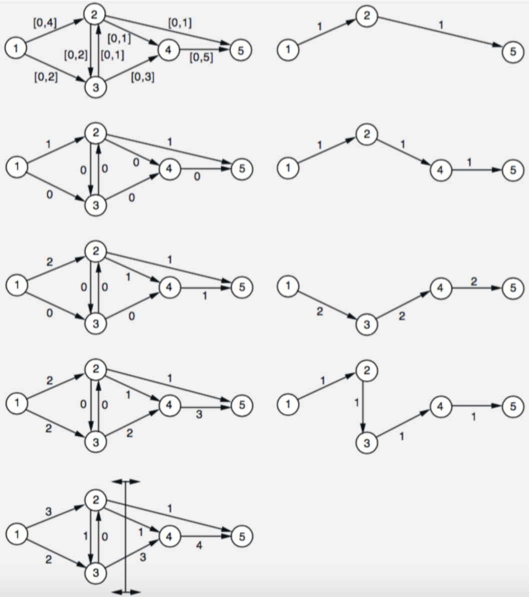
- The Ford-Fulkerson algorithm exploits the *unblocked path search algorithm*, which terminates with either (1) a simple path from s to t that is unblocked w.r.t. a given x **or** (2) a saturated cut $\mathcal{Q}$ that separates s from t.

ALGORITHM: The Ford-Fulkerson algorithm

starting with a capacity-feasible x (e.g., $x_{ij} = 0$, if $l_{ij} = 0$)
repeat
 1. Use the unblocked path search method.
 2. **if** an unblocked path P $(s \rightarrow t)$ w.r.t. x is found, **then** augment P.
until a saturated cut separating s from t is found.

- At each augmentation, the algorithm improves the primal cost by δ.
- Thus, if $\delta \geq D > 0$, the algorithm can execute only a finite number of iterations.

Example



- Let's consider again the general nonlinear network problem

$$\begin{aligned} &\text{minimize} \;\; f(x) \\ &\text{subject to} \;\; x \in F \end{aligned}$$

where F is a feasible set of the form

$$F = \left\{ x \in X \mid \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji} = s_i, \; \forall i \in \mathcal{N} \right\}$$

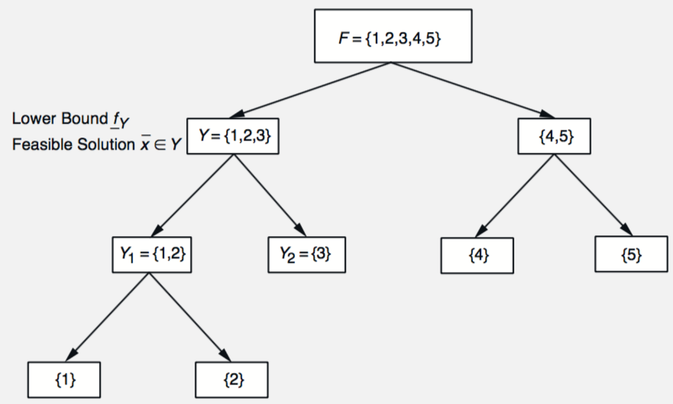
and $f : F \to \mathbf{R}$ is a given function.

- We are interested in variations of the above problem which include additional *integer contraints*.

- Some examples of such problems are:
  1. traveling salesman problem
  2. fixed charge problems (e.g., the facility section problem)
  3. minimum weight spanning tree problem
  4. matching problems (e.g., the bipartite matching problem)
  5. vehicle routing problems
  6. arc routing problems (e.g., the Chinese postman problem)

- The *branch-and-bound algorithm* implicitly enumerates all the feasible solutions, using calculations where the integer constraints are relaxed.
- Let's assume that the feasible set F is *finite*.
- The *branch-and-bound tree* is an acyclic graph, whose nodes are defined by a collection $\mathcal{F}$ of subsets of F. Specifically,

  1. $F \in \mathcal{F}$ (i.e., the set of all feasible solutions is a node).
  2. If x is feasible, then $\{x\} \in \mathcal{F}$.
  3. If $Y \in \mathcal{F}$ is not a singleton, $\exists Y_1, Y_2, \ldots, Y_n \in \mathcal{F}$ (disjoint sets) such that
  $$Y = \bigcup_{i=1}^{n} Y_i$$
  In this case, Y is said to be the *parent* of $Y_1, Y_2, \ldots, Y_n$, while the latter are called the *descendants* (or *children*) of Y.
  4. Each $Y \in \mathcal{F} \backslash \{F\}$ has a parent.



The arcs of the graph are those that connect parents Y and their children Y_i.

- **Key assumption:** There is an algorithm that calculates
  1. A lower bound $\underline{f}_Y \leq \min_{x \in Y} f(x)$, for any (nonterminal) $Y \in \mathcal{F}$.
  2. A feasible $\bar{x} \in Y$ such that $f(\bar{x})$ upper bounds the optimal cost of the original problem $\min_{x \in F} f(x)$.
- **Main idea:** Discard the nodes/branches of $\mathcal{F}$ that have no chance of containing x^*.
- To organize the search, the algorithm maintains a node list OPEN, and a scalar UPPER.
- Initially, OPEN $= \{F\}$ and UPPER $= \infty$.

ALGORITHM: Branch-and-Bound Algorithm

starting with OPEN $= \{F\}$ and UPPER $= \infty$
repeat

1. Remove a node Y from OPEN.
2. For each child Y_j in Y: **find** $\underline{f}_{Y_j}$ and a feasible $\bar{x} \in Y_j$
3. **if** $\underline{f}_{Y_j} <$ UPPER, **place** Y_j in OPEN.
4. **also if** $f(\bar{x}) <$ UPPER, **set** UPPER $= f(\bar{x})$ and "mark" $\bar{x}$.

until OPEN is empty.

- A node Y_j that is *not* placed in OPEN in Step 3 is said to be *fathomed*.
- Such a node cannot contain a better solution than the best solution $\bar{x}$ found so far.
- The algorithm is guaranteed to examine (either explicitly or implicitly) all the terminal nodes.
- Thus, it always terminates with an optimal solution.

- Branch-and-bound uses "continuous" (aka "relaxed") optimization problems to obtain the lower bounds and associated feasible solutions.
- A typical integer constraint is $x_{ij} \in \{0, 1\}$.
- In this case, a subset Y may correspond to "freezing" some values of x_{ij}, while letting the others to be either 0 or 1.
- A lower bound to $\min_{x \in Y} f(x)$ is then obtained via relaxing the 0-1 constraint on the latter x_{ij} by letting them to be $0 \leq x_{ij} \leq 1$.
- Note that the resulting problem is a convex (network) optimization problem.
- Thus, integer constraints entail the solution of many convex network problems *without* integer constraints.

- Suppose we have m clients and n locations.
- $x_{ij} = 1 \iff$ client i is assigned to location j at a cost a_{ij}.
- $y_j = 1 \iff$ a facility is placed at location j at a cost b_j.
- The problem is

$$\begin{aligned}
\min_{x,y} \quad & \sum_{(i,j)\in\mathcal{A}} a_{i,j}x_{i,j} + \sum_{j=1}^{n} b_j y_j \\
\text{subject to} \quad & \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \quad i = 1,\ldots m \\
& \sum_{\{i|(i,j)\in\mathcal{A}\}} x_{i,j} \le y_j c_j, \quad j = 1,\ldots n \\
& x_{i,j} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{A} \\
& y_j \in \{0,1\}, \quad j = 1,\ldots,n
\end{aligned}$$

where c_j is the *facility capacity* = the maximum number of customers that can be served by a facility at location j.

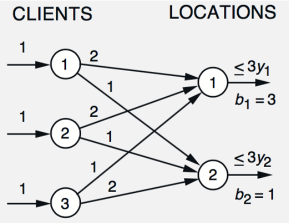
- It is convenient to select subsets of the form

$$F(J_0, J_1) = \{(x, y) \in F \mid y_j = 0, \forall j \in J_0, y_j = 1, \forall j \in J_1\}$$

where $J_0 \subset \{1, \ldots, n\}$ and $J_1 \subset \{1, \ldots, n\}$, with $J_0 \cap J_1 = \emptyset$.

- Consequently, we have "relaxed" subproblems of the form

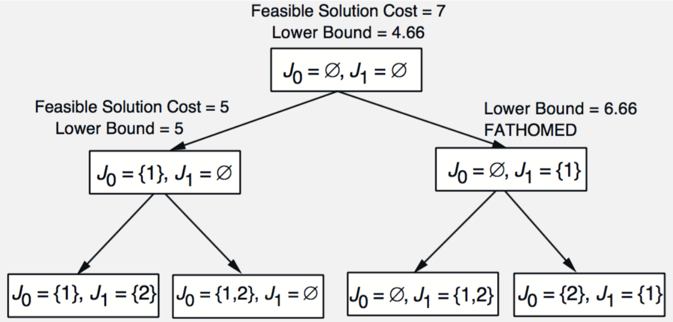
$$\begin{aligned}
\min_{x,y} \quad & \sum_{(i,j)\in\mathcal{A}} a_{i,j} x_{i,j} + \sum_{j=1}^{n} b_j y_j \\
\text{subject to} \quad & \sum_{\{j \mid (i,j)\in\mathcal{A}\}} x_{ij} = 1, \quad i = 1, \ldots m \\
& \sum_{\{i \mid (i,j)\in\mathcal{A}\}} x_{i,j} \le y_j c_j, \quad j = 1, \ldots n \\
& 0 \le x_{i,j} \le 1, \quad \forall (i,j) \in \mathcal{A} \\
& 0 \le y_j \le 1, \quad j \notin J_0 \cup J_1 \\
& y_j = 0, \quad j \in J_0 \\
& y_j = 1, \quad j \in J_1
\end{aligned}$$



- In this problem: $m = 3$, $n = 2$, $y_1^* = 0$, $y_2^* = 1$, and $f^* = 5$.
- The algorithm is initiated with the top node at $J_0 = J_1 = \emptyset$.

$$\begin{aligned}
\min_{x,y} \quad & (2x_{11} + x_{12}) + (2x_{21} + x_{22}) + (x_{31} + 2x_{32}) + 3y_1 + y_2 \\
\text{subject to} \quad & x_{11} + x_{12} = 1, \quad x_{21} + x_{22} = 1, \quad x_{31} + x_{32} = 1 \\
\text{subject to} \quad & x_{11} + x_{21} + x_{31} \leq 3y_1, \quad x_{12} + x_{22} + x_{32} \leq 3y_2 \\
& 0 \leq x_{i,j} \leq 1, \quad \forall (i,j) \in \mathcal{A} \\
& 0 \leq y_1 \leq 1, \quad 0 \leq y_2 \leq 1
\end{aligned}$$

- The problem results in: $\underline{f}_Y = 4.66$, $y_1 = 1/3$, $y_2 = 2/3$. Therefore, a feasible solution is $\bar{y}_1 = \bar{y}_2 = 1$, and the related cost is 7 (= UPPER).



- Note that the rightmost brach is fathomed, since its corresponding optimal cost (lower bound) is larger than UPPER $= 5$.
- For this problem, x^* is

$$x^*_{ij} = \begin{cases} 1, & \text{if } (i,j) = (1,2),(2,2),(3,2) \\ 0, & \text{otherwise} \end{cases}$$

- Consider the following problem with integer constraints on the arc flows:

$$\begin{aligned} \text{minimize} \quad & a^T x \\ \text{subject to} \quad & x \in F \\ & c_t^T x \le d_t, \quad t = 1, \ldots, r \\ & x_{ij} \in X_{ij}, \quad \forall (i,j) \in \mathcal{A} \end{aligned}$$

  where X_{ij} is a finite subset of contiguous integers (e.g., $X_{ij} = \{0, 1\}$ or $X_{ij} = \{1, 2, 3, 4\}$).

- We assume that the supplies s_i ("hidden" in F) are integer.

- Thus, for $r = 0$, the problem would become a minimum cost flow problem that has integer optimal solutions.

- For this, a_{ij} do not have to be integers.

- In Lagrangian relaxation, we "eliminate" the side constraints $c_t^T x \leq d_t$ by forming
$$L(x, \lambda) = a^T x + \sum_{t=1}^{r} \lambda_t (c_t^T x - d_t)$$
where $\lambda \succeq 0$ is a vector of Lagrange multipliers.

- **Key idea:** $\forall \lambda \succeq 0$, the minimization of $L(x, \lambda)$ over
$$\tilde{F} = \{x \in F \mid x_{i,j} \in X_{i,j}\}$$
yields a lower bound to the optimal cost of the original problem.

- The lower bound $\min_{x \in \tilde{F}} L(x, \lambda)$ can be used in the branch-and-bound procedure.

- The *tightest* lower bound to the optimal cost of the original problem is obtained by solving the dual problem
$$\text{maximize} \quad g(\lambda), \quad \text{s.t. } \lambda \succeq 0$$
where $g(\lambda) = \min_{x \in \tilde{F}} L(x, \lambda)$ is the dual function.

- Consider a problem of the form:

$$\begin{aligned} &\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ &\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \\ &\qquad x_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{A} \\ &\qquad \sum_{(i,j)\in\mathcal{A}} c^t_{i,j}x_{i,j} \le d^t, \quad k = 1,\ldots,r \end{aligned}$$

- Here, a path P from s to t is optimal iff x defined by

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ belongs to } P \\ 0 & \text{otherwise} \end{cases}$$

is an optimal solution of the above problem.

- Minimization of $L(x, \lambda)$ becomes a shortest path problem w.r.t. corrected arc lengths $\hat{a}_{ij}$ given by

$$\hat{a}_{ij} = a_{ij} + \sum_{t=1}^{r} \lambda_t c_{ij}^t$$

- We can then obtain λ^* that solves $\max_{\lambda \succeq 0} g(\lambda)$ and its corresponding optimal cost/lower bound.

- We can also use λ^* to obtain a feasible solution (a path that satisfies the side constraints).

- **Important:** Lagrangian relaxation eliminates the side constraints *simultaneously* with the integer constraints (since solving $\min_{x \in \tilde{F}} L(x, \lambda)$ is a (linear) minimum cost flow problem).

- This is the main reason for the widespread use of Lagrangian relaxation in combination with branch-and-bound.

- Even if we find an optimal λ, we still have only a lower bound to the optimal cost of the original problem.
- The minimization of $L(x, \lambda)$ over $\tilde{F}$ may yield an x that violates some of the side constraints $c_t^T x - d_t \leq 0$ (some heuristics might be needed).
- The maximization of $g(\lambda)$ over $\lambda \succeq 0$ may be quite nontrivial.

However ...

- In the case of a linear cost, the dual problem can be efficiently solved by a number of algorithms, such as:
  1. subgradient methods
  2. cutting plane methods

- Local search methods are a broad and important class of heuristics for discrete optimization.

- Local search methods use the notion of a *neighbourhood* $N(x)$ of a solution x.

- **Key idea:** Given a solution x, select a successor solution $\bar{x} \in N(x)$ according to some rule. Repeat the process with $\bar{x}$ replacing x.

- Thus a local search method is characterized by:
    1. The method for choosing a starting solution.
    2. The definition of $N(x)$.
    3. The rule for selecting a successor solution.
    4. The termination criterion.

- Many local search methods are *cost improving*, which means that they stop at a *local minimum*.

- There is also a basic tradeoff between using a large $N(x)$ to diminish the difficulty with local minima, and computational burden.

- **Key idea:** Modify the current solution by "splicing" and "mutation" to obtain neighbouring solutions.
- Example: In the traveling salesman problem, $N(T)$ of a tour T may be a collection of other tours obtained by modifying a contiguous portion of T, while keeping the remainder of T intact.
- It is common to maintain a pool of solutions, which "evolve" in a Darwinian way through a "survival of the fittest" process.
- Mutation allows speculative variations of the local minima at hand.
- Recombination (aka *crossover*) aims to combine attributes of a pair of local minima.
- There is a very large number of variants of genetic algorithms, which are typically problem-dependent.

We start with a population X consisting of n feasible solutions $x_1, \ldots, x_n$.

ALGORITHM: Genetic Algorithm

starting with $X = \{x_1, \ldots, x_n\}$
repeat

1. **Local Search:** for each $x_i \in X$ find a local minimum $\bar{x}_i$ by using a local search procedure. Let $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$.
2. **Mutation:** Modify a random subset of $\bar{X}$ based on some mechanism.
3. **Crossover:** Produce a feasible solution for each pair in a random subset of pairs in $\bar{X}$ based on some mechanism.
4. **Survival:** Out of all the resulting solutions, select a subset of n elements according to some criterion.
5. Use the resulting population to start the next phase.

until a stopping criterion is met.

- *Tabu search* allows avoiding poor local minima, by occasionally accepting a worse or even infeasible $\bar{x} \in N(x)$.
- Since cost improvement is not enforced, tabu search runs the danger of cycling.
- To alleviate this problem, tabu search keeps track of recently obtained solutions in a “tabu" list.
- The tabu list may contain the attributes of recently obtained solutions rather than the solutions themselves.
- Solutions with attributes in the tabu list are forbidden from being generated (unless overridden).
- Successful implementation usually depend on problem-dependent heuristics.

- *Simulated annealing* randomizes the choice of $\bar{x} \in N(x)$ in a way that gives preference to solutions of smaller cost.
- In this way, it aims to find a *global* minimum faster than “brutal" random search methods.

ALGORITHM: Genetic Algorithm

starting with a feasible solution x, $T > 0$
repeat

1. Select by random sampling $\bar{x} \in N(x)$
2. **if** $f(\bar{x}) < f(x)$, accept $\bar{x}$, i.e., $x \Leftarrow \bar{x}$.
3. **otherwise** accept $\bar{x}$ with probability $e^{-(f(\bar{x})-f(x))/T}$ (or reject).

until a stopping criterion is met.

- T is called the *temperature* of the annealing process.
- When T is large, the probability of accepting a worse solution is $\lesssim 1$.
- It is standard to start with a large T and then reduce it gradually.

- We still consider the problem $\min_{x \in F} f(x)$, where F is finite.
- A *partial solution* is $\{x_{ij} \mid (i,j) \in S\}$, where $S \subset \mathcal{A}$.
- The *rollout algorithm* generates a sequence of partial solutions, culminating with a *complete solution* (i.e., $S = \mathcal{A}$).
- The algorithm exploits *the base heuristic* which, given a partial solution P produces a *complementary solution* $\bar{P}$ and a corresponding (complete) flow vector $x = P \cup \bar{P}$.
- The *heuristic cost of the partial solution* P is defined as

$$H(P) = \begin{cases} f(x), & \text{if } x \in F \\ \infty, & \text{otherwise} \end{cases}$$

- If P is a complete and feasible, then $H(P) = f(x)$.
- There are no restrictions on the nature of the base heuristic (e.g., an integer rounding heuristic).

The rollout algorithm enlarges a partial solution iteratively, with a few arc flows at a time.

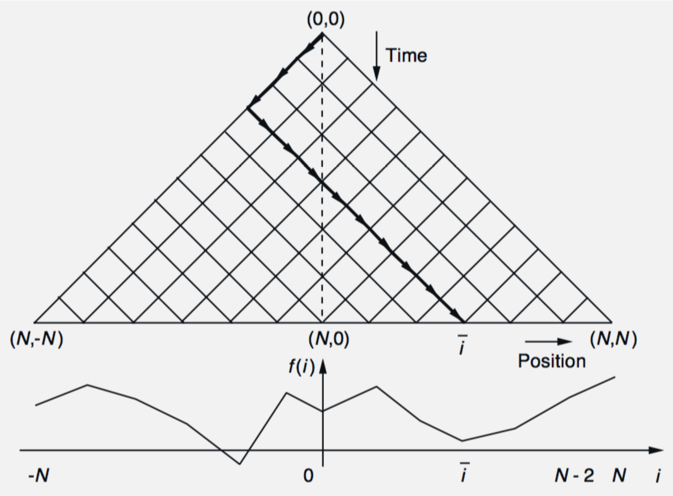
ALGORITHM: Rollout Algorithm

starting with a partial solution P with some $S \in \mathcal{A}$ (e.g., $S = \emptyset$)
repeat

1. Select $T = \{(i,j) \mid (i,j) \in \mathcal{A} \;\&\; (i,j) \notin S\}$ based on some criterion.
2. Consider the set F_T of all possible $y = \{y_{ij} \mid (i,j) \in T\}$.
3. Apply the base heuristic to compute $H(P \cup y)$ for each $y \in F_T$.
4. Choose $\bar{y} = \arg\min_{y \in F_T} H(P \cup y)$.
5. Augment P with $\bar{y}$, i.e., $P \Leftarrow P \cup \bar{y}$.

until a complete solution is obtained.

- The cost of the solutions produced by the algorithm can be shown to be monotonically nonincreasing.


(0,0)
Time
(N,-N)
(N,0)
$\bar{i}$
Position
(N,N)
f(i)
-N
0
$\bar{i}$
N-2
N
i