**EXACT SOLUTION METHODS**

* An instance: \( I = (F, c) \).
* Suppose that: \( f \in F = \{f_1, \ldots, f_n\}^T \).
* Explicit/implicit constraints.
* Examples: TSP

**SEARCH TREE EXAMPLE**

\[ f_1 = \]
\[ f_2 = \]
\[ f_3 = \]
\[ f_4 = \]
\[ f_5 = (E, F, D, C, D, E) \]
\[ f_6 = (E, F, D, E) \]
\[ f_7 = (A, A, A) \]

**EXHAUSTIVE SEARCH: BACKTRACKING**

```c
backtrack(int k)
{
    float new_cost;
    if (k == n - 1)
        new_cost := cost(val);
    if (new_cost < best_cost) {
        best_cost := new_cost;
        best_solution := copy(val);
    }
    else
        for each (el in allowed(val, k)) {
            val[k] = el;
            backtrack(k + 1);
        }
}
```

**BRANCH-AND-BOUND SEARCH**

* Partial solution: \( f^{(k)} \).
* Cost estimation of a partial solution consists of cost components for:
  + specified part of solution,
  + unspecified part of solution.

\[ c(f^{(k)}) = g(f^{(k)}) + h(f^{(k)}) \]

* In case of TSP: use spanning tree for estimation. The spanning tree is a minimal-weight tree in a graph. Consider in this case the all the points still to be interconnected; they total interconnection length will never exceed the length of the minimal spanning tree. The spanning tree can be found with a polynomial-time algorithm.
BRANCH-AND-BOUND SEARCH (CODE)

```c
b_and_b(int k)
{
    float new_cost;
    if (k == n) {
        new_cost := cost(val);
        if (new_cost < best_cost) {
            best_cost := new_cost; best_solution := copy(val);
        }
    }
    else if (lower_bound_cost(val, k) ≥ best_cost)
        /* No action, node is killed. */
    else
        for each (el ∈ allowed(val, k)) {
            val[k] := el;
            b_and_b(k + 1);
        }
}
```

BRANCH-AND-BOUND EXAMPLE

![Branch-and-Bound Example Diagram]

DYNAMIC PROGRAMMING

* Consider optimization problems characterized with a complexity parameter \( p \) (in general: multiple complexity parameters).
* Main idea: construct the optimal solution for some instance with \( p = k \) using known solutions of instances with \( p < k \); this is done by means of some construction rule.
* Use the construction rule to start building intermediate solutions starting from the smallest instances required (e.g. \( p = 0 \) or \( p = 1 \) and usually trivial to solve).
DYNAMIC PROGRAMMING FOR TSP

* Given is the graph $G(V, E)$ with edge weights $w$.
* Select an arbitrary vertex $v_s \in V$.
* $p = k$ means find shortest path from $v_s$ to any $v \in V$ that goes through exactly $k$ intermediate vertices.
* Notation: $C(S, v)$ is shortest path length from $v_s$ to $v$ exactly passing through the vertices in $S$.
* Solution amounts to computing: $C(V \setminus \{v_s\}, v_s)$.
* Construction rule:
  
  $$C(S, v) = \min_{m \in S} [C(S \setminus \{m\}, m) + w((m, v))]$$

INTEGER LINEAR PROGRAMMING (ILP)

* Special case of linear programming.
* General method to convert a large class of combinatorial optimization problems into a uniform mathematical form.
* After conversion, the problem can be solved by ILP-solvers.
* ILP is NP-complete.
* Applications in combinatorial optimization:
  + for small problem instances
  + to have certainty about exact solution for benchmarking heuristics
  + as a source of inspiration for developing new heuristics.

LINEAR PROGRAMMING EXAMPLE

* A company produces two products $P_1$ and $P_2$ with ingredients $I_1$ and $I_2$.
* $P_1$ uses $a_{11}$ units of $I_1$ and $a_{21}$ units of $I_2$. Its unit price is $c_1$. Its daily production is $x_1$ units.
* $P_2$ uses $a_{12}$ units of $I_1$ and $a_{22}$ units of $I_2$. Its unit price is $c_2$. Its daily production is $x_2$ units.
* The company cannot receive more than $b_1$ units of $I_1$ and $b_2$ units of $I_2$ per day.
* Problem: maximize the daily revenue $c_1 x_1 + c_2 x_2$ subject to
  
  $$a_{11} x_1 + a_{12} x_2 \leq b_1, \quad x_1 \geq 0$$
  $$a_{21} x_1 + a_{22} x_2 \leq b_2, \quad x_2 \geq 0$$

LINEAR PROGRAMMING (LP)

Given : matrix A vectors b, c (constants) and the vector x (unknowns).

**Canonical form:**

* Minimize or maximize:
  
  $$c^T x$$

* Subject to:
  
  $$Ax \leq b$$
  $$x \geq 0$$

**Standard form:**

* Minimize or maximize:
  
  $$c^T x$$

* Subject to:
  
  $$Ax = b$$
  $$x \geq 0$$

* The two forms can be converted into each other.
* Solvable in polynomial time by ellipsoid algorithm; in practice better performance with simplex algorithm (exponential time complexity).
INTEGER LINEAR PROGRAMMING

* Additional constraint on linear programming: all variables are integers.
* Solving the LP version first and then rounding results may give bad or unfeasible solutions.
* A special case that is often encountered is zero-one ILP: all variables can be either 0 or 1.

ILP FOR TSP

* Given is the graph $G(V, E)$ with edge weights $w$.
* Introduce a variable $x_i$ for each edge $e_i \in E$, $1 \leq i \leq k$.
* $x_i = 1$ if and only if $e_i$ is part of the solution.
* Cost function to minimize:
  $$\sum_{i=1}^{k} w(e_i)x_i$$
* Constraints to ensure:
  + that only two edges per vertex are selected;
  + that there are no multiple disjoint tours.