3-12 分支定界和局部搜索算法

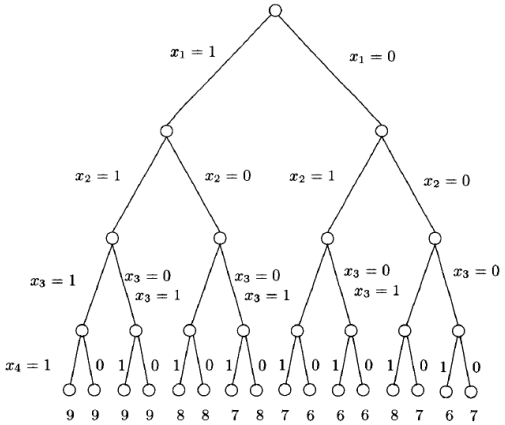
程龚

你能解释这些概念吗？

- Structure in $M(x)$

We define $\boldsymbol{T_{\mathcal{M}(x)}}$ as a labeled rooted tree with the following properties:

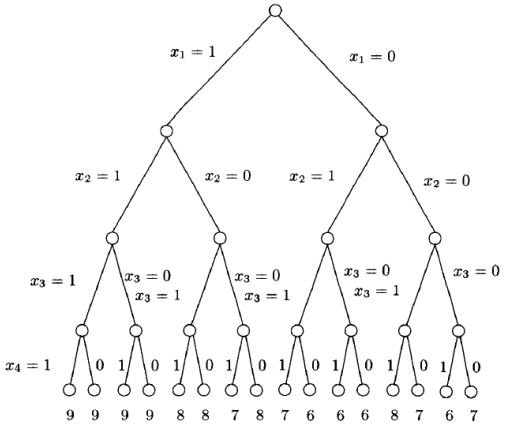
(i) Every vertex v of $T_{\mathcal{M}(x)}$ is labeled by a set $S_v \subseteq \mathcal{M}(x)$.
(ii) The root of $T_{\mathcal{M}(x)}$ is labeled by $\mathcal{M}(x)$.
(iii) If $v_1, \ldots, v_m$ are all sons of a father v in $T_{\mathcal{M}(x)}$, then $S_v = \bigcup_{i=1}^{m} S_{v_i}$ and $S_{v_i} \cap S_{v_j} = \emptyset$ for $i \neq j$.
{The sets corresponding to the sons define a partition of the set of their father.}
(iv) For every leaf u of $T_{\mathcal{M}(x)}$, $|S_u| \leq 1$.
{The leaves correspond to the feasible solutions of $\mathcal{M}(x)$.}



你能解释这些概念吗？

- Backtracking

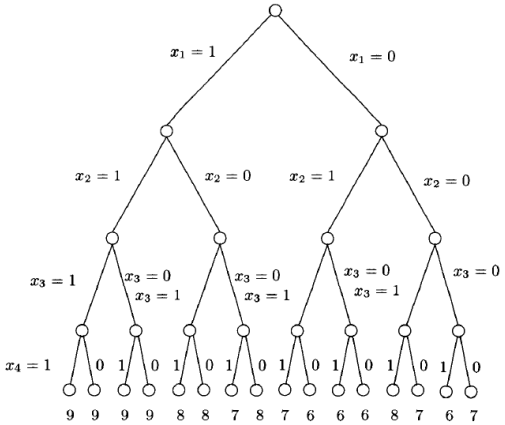
Having the tree $T_{\mathcal{M}(x)}$ the backtrack method is nothing else than a search (the depth-first-search or the breadth-first-search) in $T_{\mathcal{M}(x)}$. In fact, one does not implement backtracking as a two-phase algorithm, where $T_{\mathcal{M}(x)}$ is created in the first phase, and the second phase is the depth-first-search in $T_{\mathcal{M}(x)}$. This approach would require a too large memory. The tree $T_{\mathcal{M}(x)}$ is considered only hypothetically and one starts the depth-first-search in it directly. Thus, it is sufficient to save the path from the root to the actual vertex only.



你能解释这些概念吗？

- Branch-and-bound

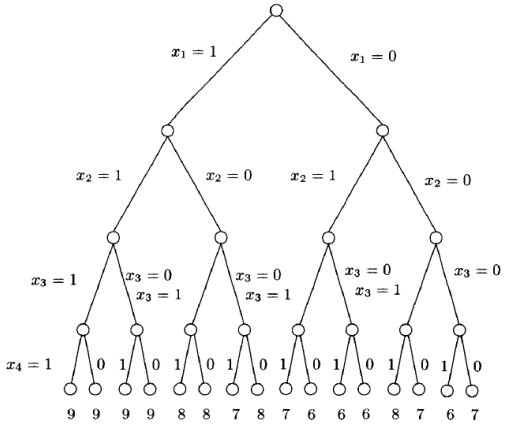
Branch-and-bound is nothing else than cutting T_v from $T_{\mathcal{M}(x)}$ if the algorithm is able to determine at the moment when v is visited (generated) that T_v does not contain any optimal solutions. The efficiency of this approach depends on the amount and the sizes of the subtrees of $T_{\mathcal{M}(x)}$ that may be cut during the execution of the algorithm.

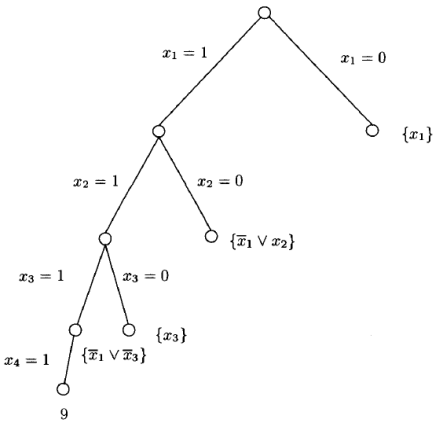


你理解这个算法了吗？

- MAX-SAT的branch-and-bound
  - 如何branch？如何bound？

$$\Phi(x_1, x_2, x_3, x_4) = (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2)$$
$$\wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$$
$$\wedge x_3 \wedge (x_1 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge x_1.$$

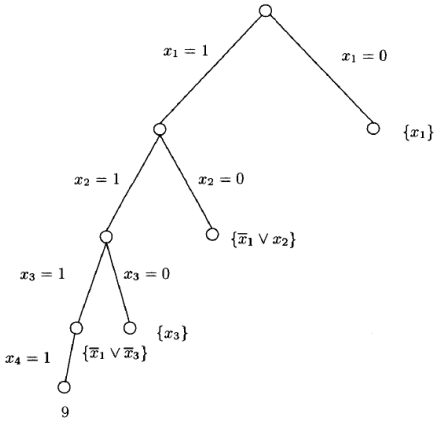


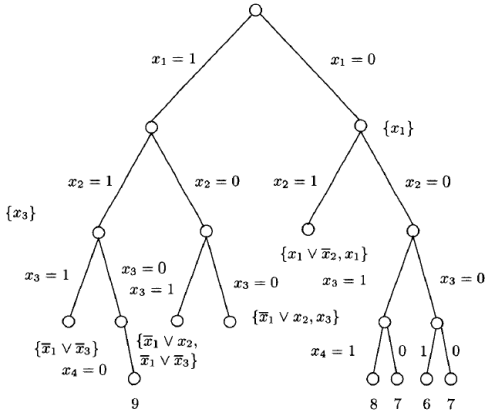


你理解这个算法了吗？

- Branch-and-bound的效率与哪些设计因素有关？

$$\Phi(x_1, x_2, x_3, x_4) = (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2)$$
$$\wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$$
$$\wedge x_3 \wedge (x_1 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge x_1.$$

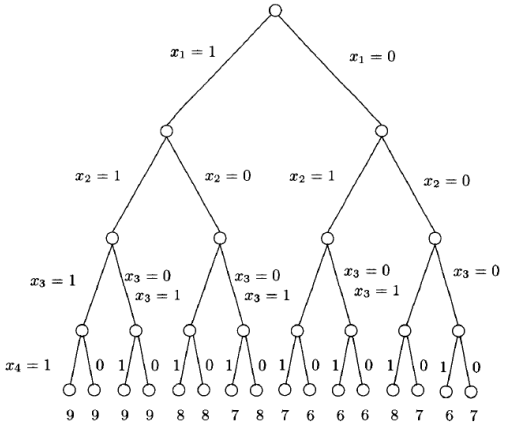




你理解这个算法了吗？

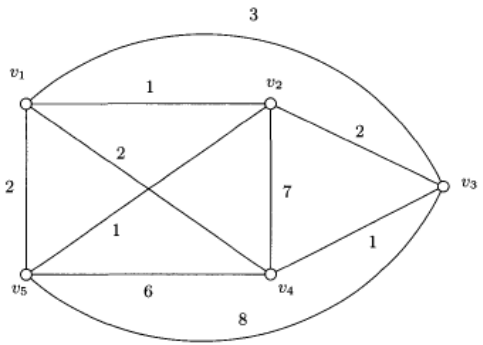
- Branch-and-bound的效率与哪些设计因素有关？
- 你能为MAX-SAT设计一种新的branch-and-bound算法吗？

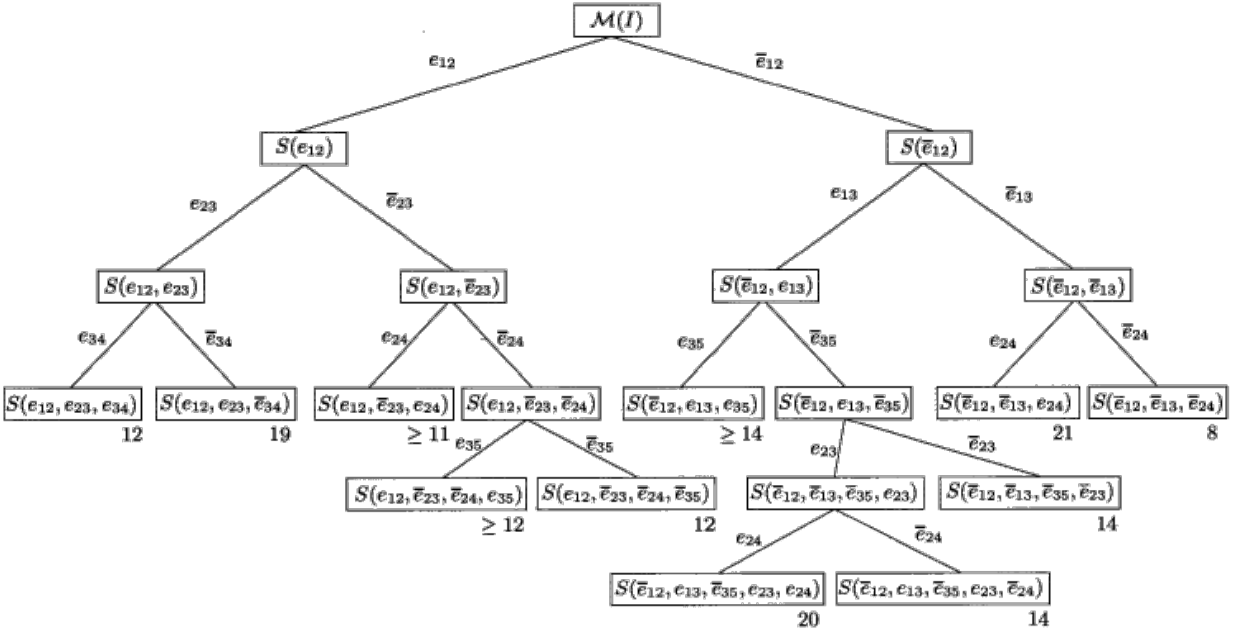
$$\Phi(x_1, x_2, x_3, x_4) = (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2)$$
$$\wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$$
$$\wedge x_3 \wedge (x_1 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge x_1.$$



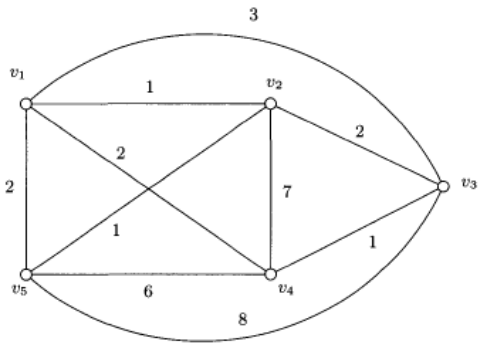
你理解这个算法了吗？

- TSP的branch-and-bound
  - 如何branch？如何bound？





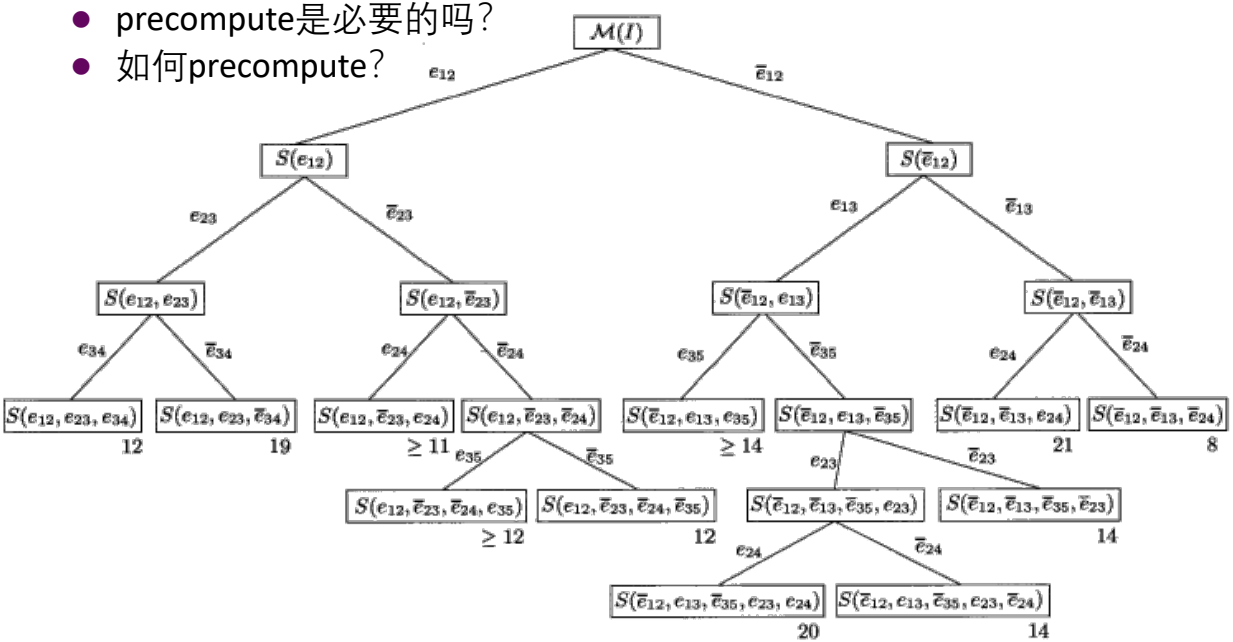
你理解这个算法了吗？



- TSP的branch-and-bound

Figure 3.11 shows the application of this branch-and-bound strategy for the instance I of TSP depicted in Figure 3.10, starting with a precomputed upper bound 10 on the optimal cost. As introduced in Section 2.3.4 $S_I(h_1, \ldots, h_r,$

- precompute是必要的吗？
- 如何precompute？



你能为以下问题设计branch-and-bound算法吗?

- MS
- SCP
- MAX-CL
- MAX-CUT
- KP
- BIN-P

你能解释这个算法的时间复杂度吗？

Assuming P $\neq$ NP, it is clear that the branch-and-bound method with any clever polynomial-time algorithmic precomputation of bounds on the cost of the optimal solutions cannot provide any polynomial-time branch-and-bound algorithm for a given NP-hard problem. The only profit that one can expect from this method is that it works within reasonable time for several of the problem instances appearing in the concrete applications.

Also, if one can precompute very good bounds by some approximation algorithm, there may exist input instances that have exponentially many feasible solutions with their costs between the optimal cost and the precomputed bound. Obviously, this may lead to a high exponential time complexity of the branch-and-bound method.

你理解这几段讨论了吗？

Since there is no general strategy for searching in $T_{\mathcal{M}(x)}$ that would be the best strategy for every input instance of a hard problem, one has to concentrate on the following two parts of the branch-and-bound method in order to increase its efficiency:

(i) to search for a good algorithm for getting bounds that are as close as possible to $Opt_U(I)$ for any input instance I of the optimization problem U, and

(ii) to find a clever, efficient strategy to compute the bounds on the costs of the solutions in the set of solutions assigned to the vertices of $T_{\mathcal{M}(I)}$.

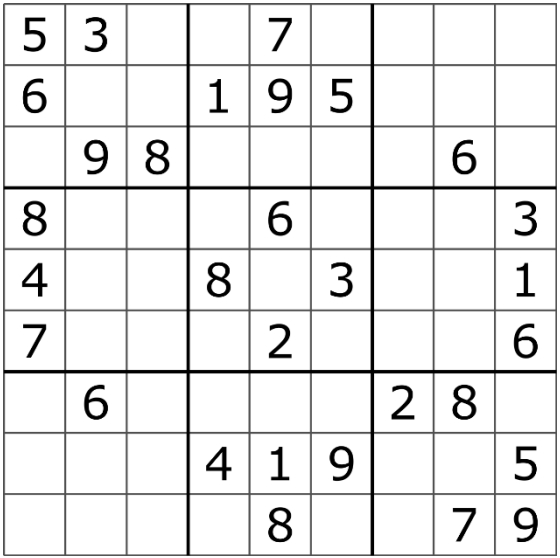
One can be willing to invest more time in searching for a good bound on $Opt_U(I)$ (because the derived bound could be essential for the complexity of the run of the most complex part (the search in $T_{\mathcal{M}(I)}$)), whereas the procedure of the estimation of a bound on a given set of solutions must be very efficient because it is used in every generated vertex, and the number of generated vertices is usually large.

Branch-and-bound思想可以用于解决判定问题吗?

- 你能举个例子吗?

Branch-and-bound思想可以用于解决判定问题吗?

- 你能举个例子吗?



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

https://en.wikipedia.org/wiki/Sudoku#/media/File:Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg

你能解释这些概念吗？

- Neighborhood

Definition 3.6.1.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem. For every* $x \in L_I$, *a* **neighborhood on** $\mathcal{M}(x)$ *is any mapping* $f_x : \mathcal{M}(x) \rightarrow Pot(\mathcal{M}(x))$ *such that*

(i) $\alpha \in f_x(\alpha)$ *for every* $\alpha \in \mathcal{M}(x)$,
(ii) if $\beta \in f_x(\alpha)$ *for some* $\alpha \in \mathcal{M}(x)$, *then* $\alpha \in f_x(\beta)$, *and*
(iii) for all $\alpha, \beta \in \mathcal{M}(x)$ *there exists a positive integer* k *and* $\gamma_1, \ldots, \gamma_k \in \mathcal{M}(x)$ *such that* $\gamma_1 \in f_x(\alpha)$, $\gamma_{i+1} \in f_x(\gamma_i)$ *for* $i = 1, \ldots, k-1$, *and* $\beta \in f_x(\gamma_k)$.

If $\alpha \in f_x(\beta)$ *for some* $\alpha, \beta \in \mathcal{M}(x)$, *we say that* α *and* β *are* **neighbors in** $\mathcal{M}(x)$. *The set* $f_x(\alpha)$ *is called the* **neighborhood of the feasible solution** α **in** $\mathcal{M}(x)$. *The (undirected) graph*

$$G_{\mathcal{M}(x), f_x} = (\mathcal{M}(x), \{\{\alpha, \beta\} \mid \alpha \in f_x(\beta), \alpha \neq \beta, \alpha, \beta \in \mathcal{M}(x)\})$$

is the **neighborhood graph of** $\mathcal{M}(x)$ **according to the neighborhood** f_x.

Let, for every $x \in L_I$, f_x *be a neighborhood on* $\mathcal{M}(x)$. *The function* $f : \bigcup_{x \in L_I}(\{x\} \times \mathcal{M}(x)) \rightarrow \bigcup_{x \in L_I} Pot(\mathcal{M}(x))$ *with the property* $f(x, \alpha) = f_x(\alpha)$ *for every* $x \in L_I$ *and every* $\alpha \in \mathcal{M}(x)$ *is called a* **neighborhood for** U.

你能解释这些概念吗?

- Neighborhood graph
  - 这个图具有哪些性质?
- Local transformation
  - 你能举个例子吗?
- Local optimum

你能解释这些概念吗？

- Neighborhood graph
  - 这个图具有哪些性质？
- Local transformation
  - 你能举个例子吗？
- Local optimum

Definition 3.6.1.4. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem, and let, for every* $x \in L_I$*, the function* f_x *be neighborhood on* $\mathcal{M}(x)$*. A feasible solution* $\alpha \in \mathcal{M}(x)$ *is a* **local optimum for the input instance** x **of** U **according to** f_x*, if*

$$cost(\alpha) = goal\{cost(\beta) \mid \beta \in f_x(\alpha)\}.$$

你理解这个算法了吗？

LSS(*Neigh*)-Local Search Scheme according to a neighborhood *Neigh*

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

你理解这个算法了吗？

LSS(*Neigh*)-Local Search Scheme according to a neighborhood *Neigh*

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

Theorem 3.6.1.5. *Any local search algorithm based on* LSS(*Neigh*) *for an optimization problem U outputs, for every instance x of U, a local optimum for x according to the neighborhood Neigh.*

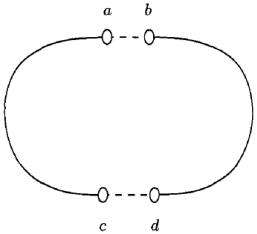
你理解这个算法了吗？

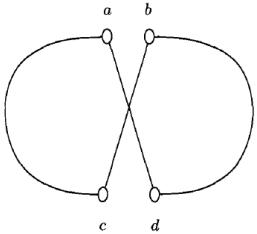
LSS($Neigh$)-Local Search Scheme according to a neighborhood $Neigh$

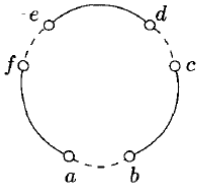
Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

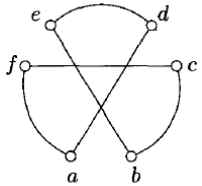
- Local search的质量/效率与哪些设计因素有关？

你理解这两个算法了吗?

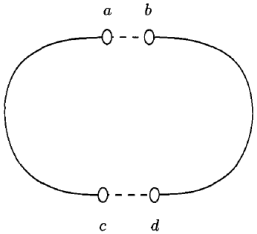


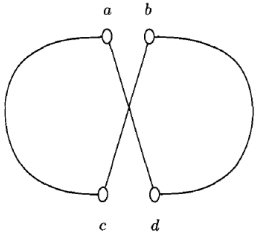




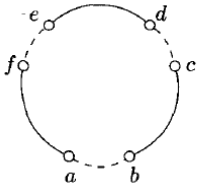


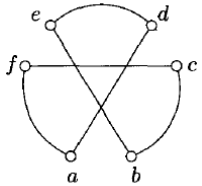
你理解这两个算法了吗？



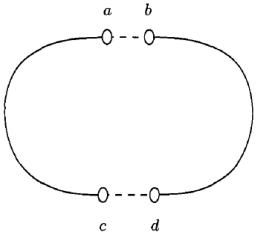


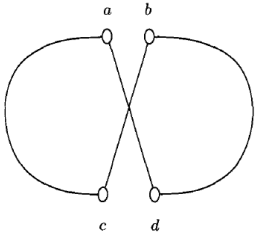
the size of the neighborhood 2-*Exchange*(α) is $\frac{n\cdot(n-3)}{2}$

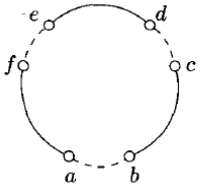


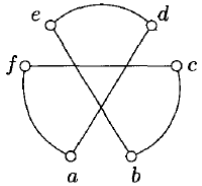


你理解这两个算法了吗？









- 你能比较它们的优劣吗？

你能为以下问题定义neighborhood吗？

- MAX-SAT
- MAX-CUT

- MS
- SCP
- MAX-CL
- KP
- BIN-P

你能解释这个算法的时间复杂度吗？

LSS($Neigh$)-Local Search Scheme according to a neighborhood $Neigh$

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

Theorem 3.6.1.6. *Let $U = (\Sigma_O, \Sigma_I, L, L_I, \mathcal{M}, cost, goal)$ be an integer-valued optimization problem with the cost function cost from feasible solutions to positive integers. Let there exist a polynomial p such that $cost(\alpha, x) \leq p(Max\text{-}Int(x))$ for every $x \in L_I$ and every $\alpha \in \mathcal{M}(x)$. For every neighborhood Neigh such that $Neigh_x(\alpha)$ can be generated from α and x in polynomial time in $|x|$ for every $x \in L_I$ and every $\alpha \in \mathcal{M}(x)$, $LSS(Neigh)$ provides a pseudo-polynomial-time algorithm that finds a local optimum according to the neighborhood Neigh.*

你理解这段讨论了吗？

neighborhood. If a neighborhood *Neigh* has the property that $Neigh(\alpha)$ has a small cardinality for every $\alpha \in \mathcal{M}(x)$, then one iterative improvement of Step 2 of LSS(*Neigh*) can be executed efficiently but the risk that there are many local optima (potentially with a cost that is very far from $Opt_U(x)$) can substantially grow. On the other hand, large $|Neigh_x(\alpha)|$ can lead to feasible solutions with costs that are closer to $Opt_U(x)$ than small neighborhoods can, but the complexity of the execution of one run of the while cycle in Step 2 can increase too much. Thus, the choice of the neighborhood is always a game with the tradeoff between the time complexity and the quality of the solution. Small neighborhoods are typical for most applications. But a small neighborhood alone does not provide any assurance of the efficiency of the local search algorithm, because the number of runs of the while cycle of the local search scheme can be exponential in the input size. We can only guarantee a

你理解这个算法了吗？

KL(*Neigh*) Kernighan-Lin Variable-Depth Search Algorithm with respect to the neighborhood *Neigh*

Input: An input instance I of an optimization problem U.

Step 1: Generate a feasible solution $\alpha = (p_1, p_2, \ldots, p_n) \in \mathcal{M}(I)$ where $(p_1, p_2, \ldots, p_n)$ is such a parametric representation of α that the local transformation defining *Neigh* can be viewed as an exchange of a few of these parameters.

Step 2: $IMPROVEMENT := TRUE$;
$EXCHANGE := \{1, 2, \ldots, n\}$; $J := 0$; $\alpha_J := \alpha$;
while $IMPROVEMENT = TRUE$ **do begin**
  while $EXCHANGE \neq \emptyset$ **do**
    begin $J := J + 1$;
      $\alpha_J :=$ a solution from $Neigh(\alpha_{J-1})$ such that $gain(\alpha_{J-1}, \alpha_J)$ is the maximum of $\{gain(\alpha_{J-1}, \delta) \mid \delta \in Neigh(\alpha_{J-1}) - \{\alpha_{J-1}\}$ and δ differs from α_{J-1} in the parameters of $EXCHANGE$ only$\}$;
      $EXCHANGE := EXCHANGE - \{$the parameters in which α_J and α_{J-1} differ$\}$
  end;
  Compute $gain(\alpha, \alpha_i)$ for $i = 1, \ldots, J$;
  Compute $l \in \{1, \ldots, J\}$ such that

$$gain(\alpha, \alpha_l) = \max\{gain(\alpha, \alpha_i) \mid i \in \{1, 2, \ldots, J\}\};$$

  if $gain(\alpha, \alpha_l) > 0$ **then**
    begin $\alpha := \alpha_l$;
      $EXCHANGE := \{1, 2, \ldots, n\}$
    end
  else $IMPROVEMENT := FALSE$
end

Step 3: **output**(α).

α. The main idea of the above-described approach is that a few steps in the wrong direction (when $gain(\alpha, \alpha_1), gain(\alpha_1, \alpha_2), \ldots, gain(\alpha_r, \alpha_{r+1})$ are all negative) may ultimately be redeemed by a large step in the right direction ($gain(\alpha_r, \alpha_{r+1}) \gg |\sum_{i=0}^{r} gain(\alpha_i, \alpha_{i+1})|$, for instance).

你理解这个算法了吗？

- In numerical analysis, **hill climbing** is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.

- α、*Neigh*、β?
- 你能以TSP为例，给出一个具体的算法吗？
- 它够好了吗？

LSS(*Neigh*)-Local Search Scheme according to a neighborhood *Neigh*

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

你理解这个算法了吗？

- For a **very large-scale neighborhood search**, the neighborhood is large and possibly exponentially sized.

- 相比hill climbing，改变了α、$Neigh$、β?
- 你能以TSP为例，给出一个具体的算法吗？
- 它够好了吗？

LSS($Neigh$)-Local Search Scheme according to a neighborhood $Neigh$

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

你理解这个算法了吗？

- **Multi-start methods**: re-start the procedure from a new solution once a region has been explored.

- 相比hill climbing，改变了α、$Neigh$、β?
- 你能以TSP为例，给出一个具体的算法吗？
- 它够好了吗？

LSS($Neigh$)-Local Search Scheme according to a neighborhood $Neigh$

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

你理解这个算法了吗？

- **Stochastic hill climbing** chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move.

- 相比hill climbing，改变了α、$Neigh$、β?
- 你能以TSP为例，给出一个具体的算法吗？
- 它够好了吗？

LSS($Neigh$)-Local Search Scheme according to a neighborhood $Neigh$

Input: An input instance x of an optimization problem U.
Step 1: Find a feasible solution $\alpha \in \mathcal{M}(x)$.
Step 2: **while** $\alpha \notin LocOPT_U(x, Neigh_x)$ **do**
begin find a $\beta \in Neigh_x(\alpha)$ such that
$cost(\beta) < cost(\alpha)$ if U is a minimization problem and
$cost(\beta) > cost(\alpha)$ if U is a maximization problem; $\alpha := \beta$
end
Output: **output**(α).

OT

- 除MAX-SAT和TSP外，分支定界和局部搜索算法还可用于解决其它问题，请为每种算法调研至少1种可以解决的问题，结合例子介绍算法的设计与分析。