干货 | 自适应大邻域搜索(Adaptive Large Neighborhood Search)入门到精通超详细解析-概念篇

原创 邓发珩 数据魔术师 2019-01-01

前言

各位小伙伴大家好呀~最近好久没有给大家推过干货了，不过小编可没有闲着。最近一直在苦苦研究的**neighborhood search**终于有了结果。

至于这是什么东西呢？咳咳，这可不是一起去你邻居家找茬的啊……

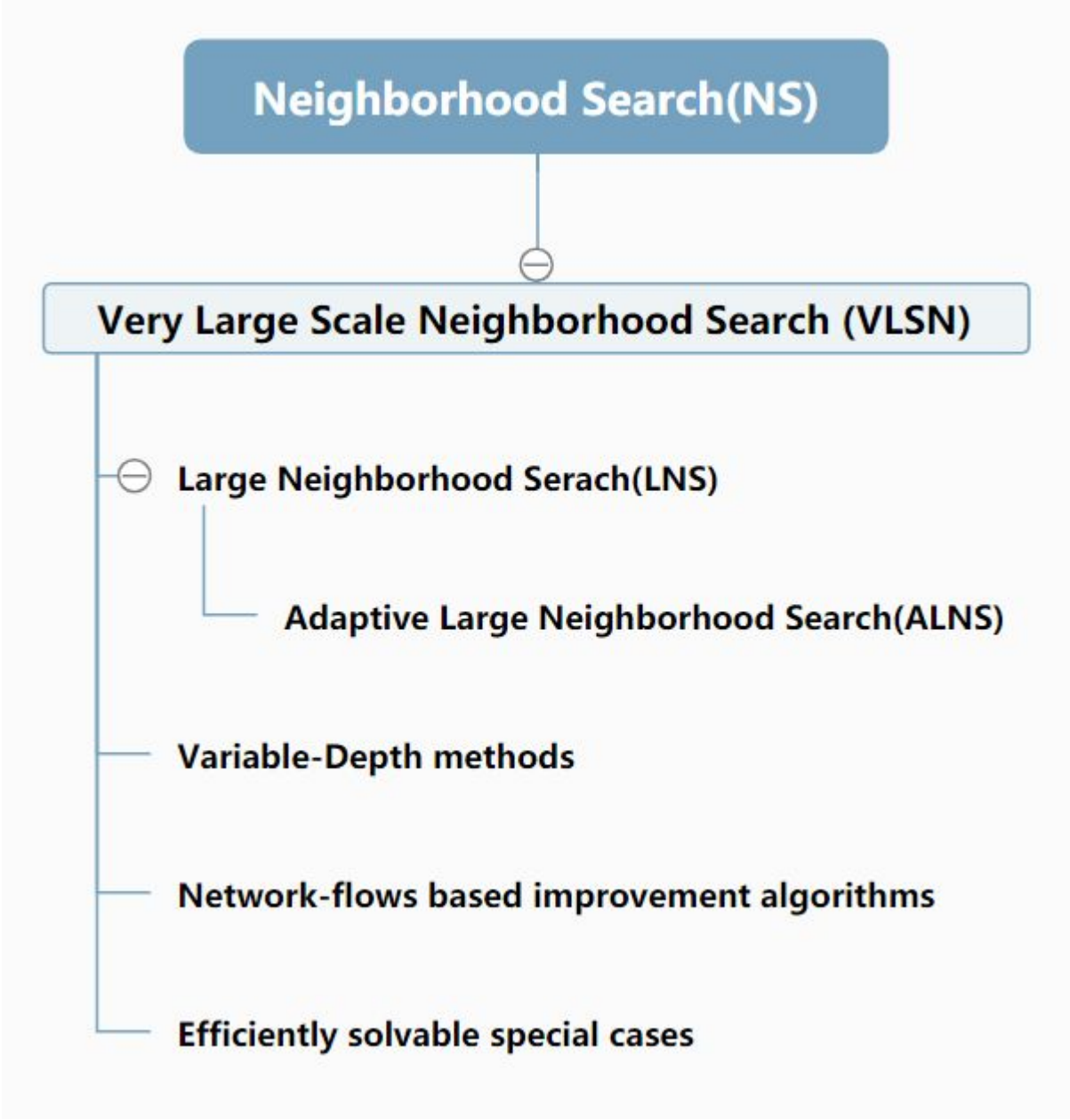
言归正传，关于大邻域搜索这一块的启发式算法，国内还算比较少的了。那么，今天和小编一起，一步一个脚印来了解这个神秘的算法吧。

01 概念科普篇

关于neighborhood serach，这里有好多种**衍生**和**变种**出来的胡里花俏的算法。大家在上网搜索的过程中可能看到什么Large Neighborhood Serach，也可能看到Very Large Scale Neighborhood Search或者今天介绍的Adaptive Large Neighborhood Search。

对于这种名字相近，实则大有不同的概念，很是让小编这样的新手头疼。不过，小编喜欢凡事都要弄得清清楚楚明明白白的。为了防止大家混淆这些相近的概念，今天这里一并介绍了吧。

总体关系可以看下图



当一个邻域搜索算法搜索的邻域规模随着算例规模的增大而呈指数增长，或者邻域太大而不能在实际中明确搜索时，我们把这类邻域搜索算法归类为Very Large-Scale Neighborhood Search(VLSN)。

VLSN又可以分为三类：[1]

- **Variable-depth methods**
- **Network-flows based improvement algorithms**
- **Efficiently solvable special cases**

而Large Neighborhood Search(LNS) 则不属于以上三种类型，**但它确实是属于VLSN这种类型的，因为它搜索的是一个非常大的邻域。**

最后呢，是Adaptive Large Neighborhood Search(ALNS)，它是根据Large Neighborhood Search(LNS) **算法扩展和延伸而来**（嗯，相当于爸爸和儿子的关系……）。

由于文章篇幅呢，小编这里就不给大家一一介绍了。具体内容可以看文章后面给出的参考文献。**下面给大家科普几个必要的概念。**

1.0 什么是Neighborhood Search(NS)

邻域搜索算法（或称为局部搜索算法）是一类非常常见的改进算法，其在每次迭代时通过搜索当前解的"邻域"找到更优的解。 **邻域搜索算法设计中的关键是邻域结构的选择，即邻域定义的方式。** 根据以往的经验，邻域越大，局部最优解就越好，这样获得的全局最优解就越好。 但是，与此同时，邻域越大，每次迭代搜索邻域所需的时间也越长。**出于这个原因，除非能够以非常有效的方式搜索较大的邻域，否则启发式搜索也得不到很好的效果。[2]**

什么又是**邻域**呢?

小编不得不再次带大家回顾一下以前的知识：

官方一点：所谓邻域，简单的说即是给定点附近其它点的集合。在距离空间中，邻域一般被定义为以给定点为圆心的一个圆；而在组合优化问题中，邻域一般定义为由给定转化规则对给定的问题域上每结点进行转化所得到的问题域上结点的集合 （太难懂了 呜呜呜.....）。

通俗一点：邻域就是指对当前解进行一个操作(这个操作可以称之为邻域动作)可以得到的所有解的集合。那么不同邻域的本质区别就在于邻域动作的不同了。

邻域动作又是什么鬼?

没关系，咱们再回顾一下：

邻域动作是一个函数，通过这个函数，对当前解s，产生其相应的邻居解集合。例如：对于一个bool型问题，其当前解为：s = 1001，**当将邻域动作定义为翻转其中一个bit时**，得到的邻居解的集合N(s)={0001,1101,1011,1000}，其中N(s) ∈ S。同理，当将邻域动作定义为互换相邻bit时，得到的邻居解的集合N(s)={0101,1001,1010}。

碍于文章篇幅，小编就不再对Neighborhood Search深入介绍了。不过小编给大家找了一个比较详细的定义，大家可以看看：[1]

Neighborhood search

In this section we formally introduce the term neighborhood search. We are given an instance I of a combinatorial optimization problem where X is the set of feasible solutions for the instance (we write $X(I)$ when we need to emphasize the connection between instance and solution set) and $c : X \rightarrow \mathbb{R}$ is a function that maps from a solution to its *cost*. X is assumed to be finite, but is usually an extremely large set. We assume that the combinatorial optimization problem is a minimization problem, that is, we want to find a solution x^* such that $c(x^*) \leq c(x)\, \forall x \in X$.

We define a *neighborhood* of a solution $x \in X$ as $N(x) \subseteq X$. That is, N is a function that maps a solution to a set of solutions. A solution x is said to be *locally optimal* or a *local optimum* with respect to a neighborhood N if $c(x) \leq c(x')\, \forall x' \in N(x)$. With these definitions it is possible to define a neighborhood search algorithm. The algorithm takes an initial solution x as input. It computes $x' = \arg\min_{x'' \in N(x)} \{c(x'')\}$, that is, it finds the cheapest solution x' in the neighborhood of x. If $c(x') < c(x)$ then the algorithm performs the update $x = x'$. The neighborhood of the new solution x is searched for an improving solution and this is repeated until a local optimum x is reached. When this happens the algorithm stops. The algorithm is denoted a *steepest descent* algorithm as it always chooses the best solution in the neighborhood.

A simple example of a neighborhood for the TSP is the *2-opt* neighborhood which can be traced back to [15]. The neighborhood of a solution x in the 2-opt neighborhood is the set of solutions that can be reached from x by deleting two edges in x and adding two other edges in order to reconnect the tour. A simple example of a neighborhood for the CVRP is the *relocate* neighborhood (see e.g. [27]). In this neighborhood, $N(x)$ is defined as the set of solutions that can be created from x by relocating a single customer. The customer can be moved to another position in its current route or to another route.

We define the size of the neighborhood $N(\cdot)$ for a particular instance I as $\max\{|N(x)| : x \in X(I)\}$. Let $\mathscr{I}(n)$ be the (possibly infinite) set of all instances of size n of the problem under study. We can then define the size of a neighborhood as a function $f(n)$ of the instance size n: $f(n) = \max\{|N(x)| : I \in \mathscr{I}(n), x \in X(I)\}$.

（图1：Neighborhood Search的详细定义）

1.1 什么是VLSN?

正如前面所说的一样，**对于一个邻域搜索算法，当其邻域大小随着输入数据的规模大小呈指数增长的时候**，那么我们就可以称该邻域搜索算法为超大规模邻域搜索算法（Very Large Scale Neighborhood Search Algorithm，VLSNA ）。

一些超大规模的邻域搜索方法已经运用于运筹学之中了，并且取得了不错的效果。 例如，如果将求解线性规划的单纯形算法看成邻域搜索算法的话，那么列生成算法就是一种超大规模的邻域搜索方法。 此外，用于解决许多网络流问题的方法也可以归类为超大规模的邻域搜索方法。 用于求解最小费用流问题的negative cost cycle canceling algorithm和用于求解分配问题的augmenting path algorithm就是两个例子。[1]

有关于VLSN我们就介绍这么多啦。不过小编还是把Wikipedia上关于VLSN的定义也放上来给大家看看吧：
https://en.wikipedia.org/wiki/Very_large-scale_neighborhood_search

1.2 什么是LNS

Large Neighborhood Serach，LNS，**是VLSN的一个实例。**大多数邻域搜索算法都明确定义它们的邻域，如在上面1.0 节图1所示。 在LNS中，邻域是由destroy和repair方法隐式定义的。**destroy方法会破坏当前解的一部分，而后repair方法会对被破坏的解进行重建。**destroy方法通常包含随机性的元素，以便在每次调用destroy方法时破坏解的不同部分。

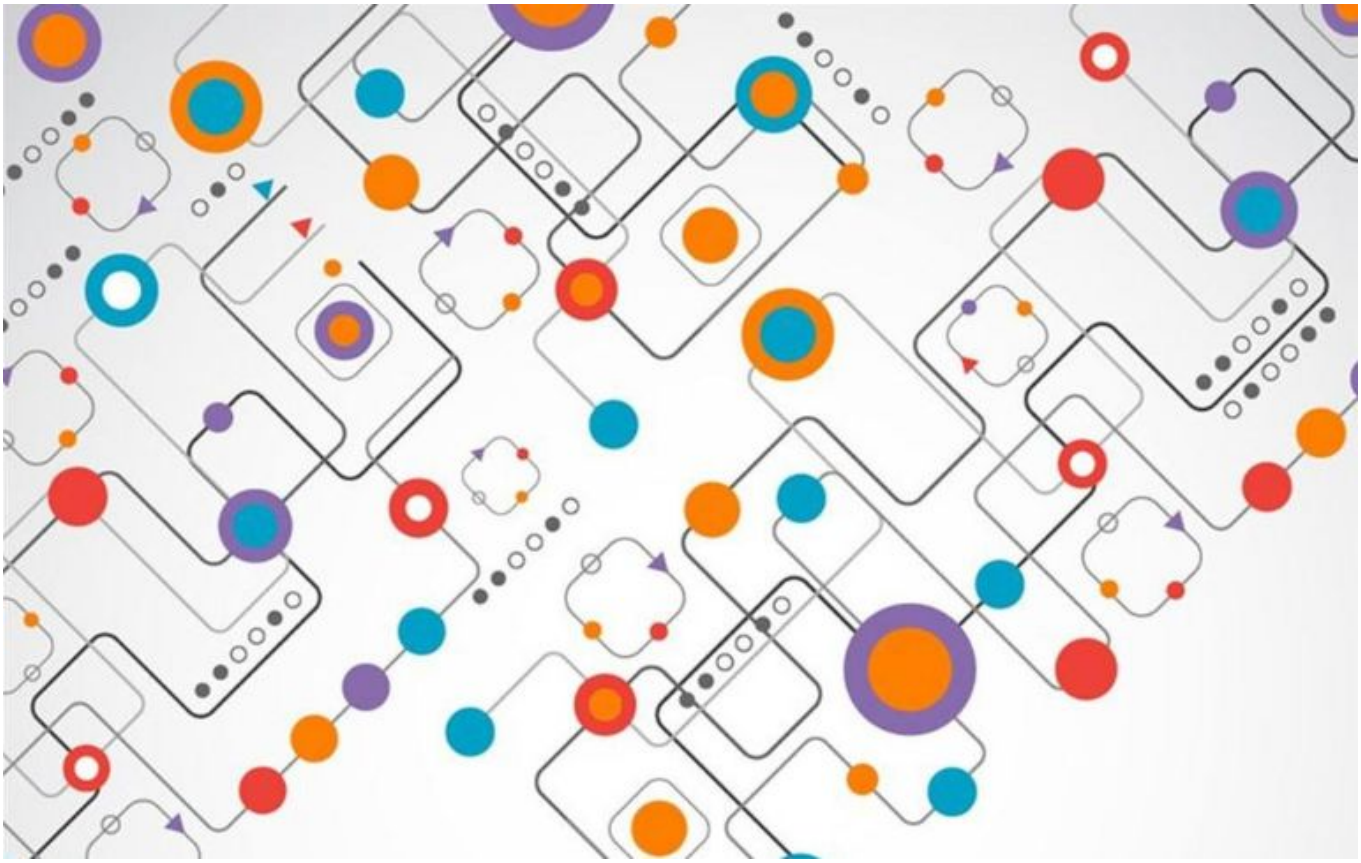
那么，解x的邻域N（x）就可以定义为：首先通过利用destroy方法破坏解x，然后利用repair方法重建解x，从而得到的一系列解的集合。

1.3 什么是ALNS

我们前面说过，**ALNS是从LNS发展扩展而来的**，在了解了LNS以后，我们现在来看看**ALNS**。**ALNS**在LSN的基础上，**允许在同一个搜索中使用多个destroy和repair方法来获得当前解的邻域。**

ALNS会为每个destroy和repair方法分配一个权重，通过该权重从而控制每个destroy和repair方法在搜索期间使用的频率。 在搜索的过程中，**ALNS**会对各个destroy和repair方法的权重进行**动态调整**，以便获得更好的邻域和解。简单点解释，**ALNS**和LNS不同的是，**ALNS通过使用多种destroy和repair方法，然后再根据这些destroy和repair方法生成的解的质量，选择那些表现好的destroy和repair方法，再次生成邻域进行搜索。**

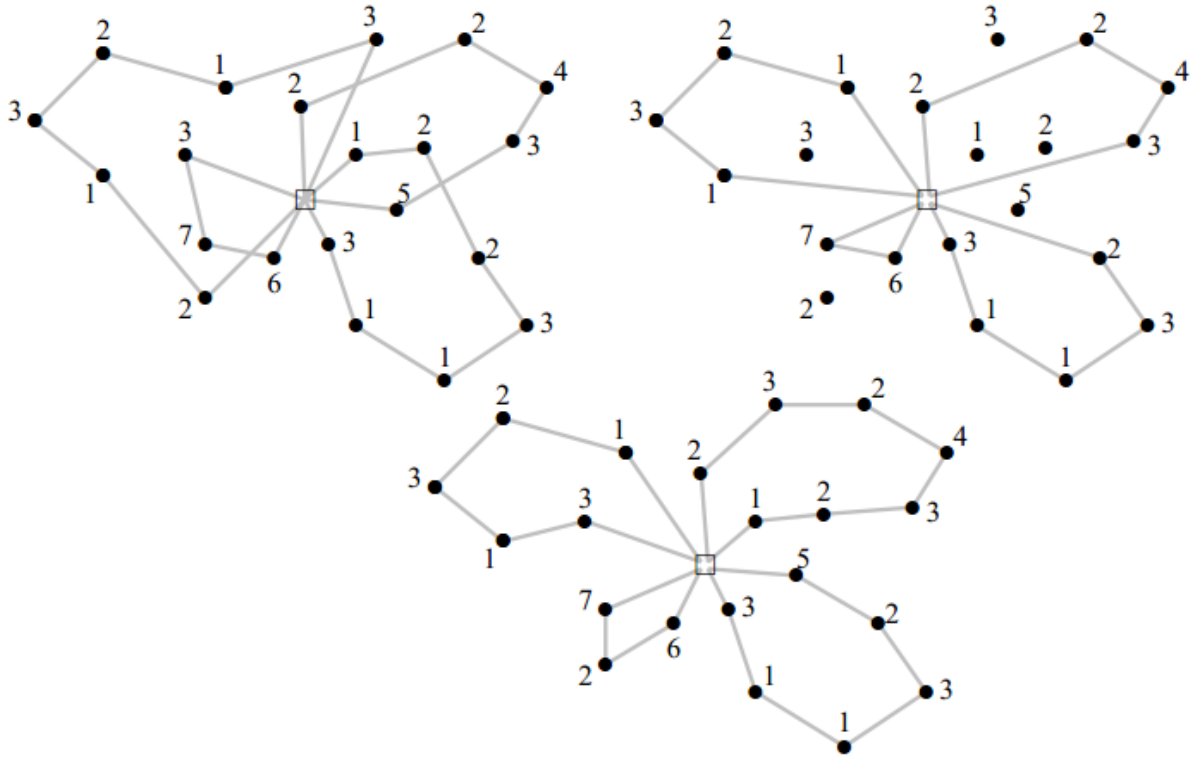
02 具体流程篇



关于Adaptive Large Neighborhood Search(ALNS)的具体搜索过程，在这一节我们来介绍一下。不过，在此之前，还是先介绍一些必要的概念。

2.1 destroy和repair方法

关于destroy和repair方法，这**两个小老弟**在LNS和**ALNS**中是要**组合**在一起使用的，待会你们就明白了。**其实，一个解x经过destroy和repair方法以后，实则是相当于经过了一个邻域动作的变换。**如下图所示：[1]



上图是三个CVRP问题的解（别问我什么是CVRP问题！），上左表示的是**当前解**，上右则是**经过了destroy方法以后的解**（移除了6个customers），下面表示由上右的解**经过repair方法以后最终形成的解**（重新插入了一些customers）。

哎哎哎!等等，小编刚刚不是说当前解x经过destroy和repair方法以后生成的是一个邻域（邻居解的集合）吗？上面才生成一个解呀！

其实，上面展示的只是生成邻域中一个解的过程而已，实际整个邻域还有很多其他可能的解。比如在一个CVRP问题中，将destroy方法定义为：移除当前解x中**15%**的customers。假如当前的解x中有**100名customers**，那么就有**C（100,15）= 100!/(15!×85!) =2.5×10的17次方** 种移除的方式。并且，根据每一种移除方式，又可以有很多种修复的方法。这样下来，一对destroy和repair方法能生成**非常多**的邻居解，**而这些邻居解的集合，就是邻域了。**

2.2 LNS的具体流程

我们说过，**ALNS是由LNS扩展而来的**，在了解**ALNS**之前，我们不妨来了解一下LNS的具体流程。下面是LNS的伪代码：[1]

Algorithm 1 Large neighborhood search

```
1:  input: a feasible solution x
2:  x^b = x;
3:  repeat
4:      x^t = r(d(x));
5:      if accept(x^t, x) then
6:          x = x^t;
7:      end if
8:      if c(x^t) < c(x^b) then
9:          x^b = x^t;
10:     end if
11: until stop criterion is met
12: return x^b
```

LNS算法中包含三个变量。变量x^b记录**目前为止获得的最优解**，x则表示**当前解**，而x^t是**临时解**（便于回退到之前解的状态）。

函数d（·）是**destroy方法**，而r（·）是**repair方法**。详细点就是，d（x）表示破坏解x的部分，而r（x）则表示对破坏的解进行重新修复。

在第2行中，初始化了全局最优解。在第4行中，算法首先用destroy方法，然后再用repair方法来获得临时解x^t。在第5行中，评估临时解x^t的好坏，并以此确定该临时解x^t是否应该成为当前新的解x（第6行）。

评估的方式因具体程序而异，可以有很多种。最简单的评估方式就只接受那些变得更优的解。注：评估准则可以参考**模拟退火的算法原理**，设置一个接受的可能性，效果也许会更佳。

第8行检查新解x是否优于全局最优解x^b。此处 c（x）表示解x的目标函数值。如果新获得的解x更优，那么第9行将会更新全局最优解x^b。在第11行中，检查终止条件。在第12行中，返回找到的全局最优解x^b。

从伪代码可以注意到，**LNS算法不会搜索解的整个邻域，而只是对该邻域进行采样搜索。也就是说，这么大的邻域是不可能一一遍历搜索的，只能采样，搜索其中的一些解而已。**

2.3 ALNS的具体流程

好了，介绍完了LNS的具体流程，终于到今天的主角**ALNS**了。在理解了LNS的基础上，理解**ALNS**也非常easy了。前面说过，**ALNS**对LNS进行了扩展，**它允许在一次搜索中搜索多个邻域（使用多组不同的destroy和repair方法）。**至于搜索哪个邻域呢，**ALNS会根据邻域解的质量好坏，动态进行选择调整。**好啦，来看伪代码：[1]

Algorithm 2 Adaptive large neighborhood search

```
1:  input: a feasible solution x
2:  x^b = x; ρ^- = (1,...,1); ρ^+ = (1,...,1);
3:  repeat
4:      select destroy and repair methods d ∈ Ω^- and r ∈ Ω^+ using ρ^- and ρ^+;
5:      x^t = r(d(x));
6:      if accept(x^t, x) then
7:          x = x^t;
8:      end if
9:      if c(x^t) < c(x^b) then
10:         x^b = x^t;
11:     end if
12:     update ρ^- and ρ^+;
13: until stop criterion is met
14: return x^b
```

上面就是ALNS伪代码。相对于LNS来说，**新增了第4行和第12行，修改了第2行。**

Ω^-和Ω^+分别表示destroy和repair方法的集合。第2行中的ρ^-和ρ^+分别表示各个destroy和repair方法的权重集合。一开始时，**所有的方法都设置相同的权重。**

第4行**根据ρ^-和ρ^+选择destroy和repair方法。**至于选择哪个方法的可能性大小，是由下面公式算出的：[1]

$$\phi_j^- = \frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-},$$

总的来说，权重越大，被选中的可能性越大。

除此之外，权重大小是根据destroy和repair方法的在搜索过程中的表现进行动态调整的（第12行）。具体是怎么调整的呢？这里也给大家说一说：

在ALNS完成一次迭代搜索以后，我们使用下面的函数为每组destroy和repair方法的好坏进行一个评估：[1]

$$\psi = \max \begin{cases} \omega_1 & \text{if the new solution is a new global best,} \\ \omega_2 & \text{if the new solution is better than the current one,} \\ \omega_3 & \text{if the new solution is accepted,} \\ \omega_4 & \text{if the new solution is rejected,} \end{cases} \quad (1)$$

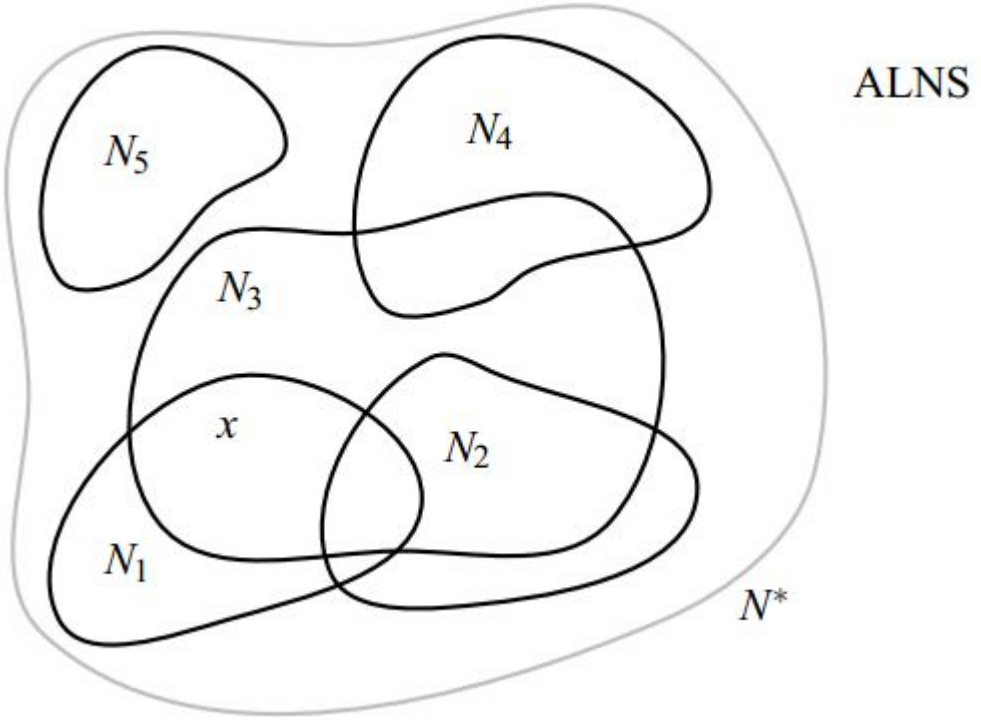
其中，ω_1≥ω_2≥ω_3≥ω_4≥0。为自定的参数。

假如，a和b是上次迭代中使用的destroy和repair方法。那么其权重更新如下所示：

$$\rho_a^- = \lambda \rho_a^- + (1-\lambda)\psi, \quad \rho_b^+ = \lambda \rho_b^+ + (1-\lambda)\psi,$$

其中，λ∈[0,1]，为参数。

再来给大家看个图



在一个ALNS算法中，有很多个邻域，**每个邻域都可以看做是一组destroy和repair方法生成的。**

03 代码实战篇

碍于文章篇幅原因，今天就先不上代码了。大家先把这些概念好好理解消化了先。后面小编会把代码和详细解释做成一篇篇文章推送给大家的。谢谢！

后记

做这篇文章，也可谓是费了九牛二虎之力。一开始被各种铺天盖地的概念搅得昏天暗地，然后搭上梯子到墙外面查找各种文献和资料，才算有了一点眉头。由于小编的理解能力有限，可能很多概念理解还不到位，解释不够清楚。也请各位看官高抬贵眼，**如有错误，欢迎随时和小编联系进行纠正。**最后，谢谢大家！元旦快乐。

参考文献

[1]"David Pisinger and Stefan Ropke",Large neighborhood search.

[2]"Ravindra K. Ahuja, Ozlem Ergun, James B. Orlin, Abraham P. Punnen", A survey of very large-scale neighborhood search techniques.

[3]"Christine Bachoc, Dion C. Gijswijt, Alexander Schrijver, and Frank Vallentin", Invariant semidefinite programs.

[4]"Mhand Hi, Ibrahim Moussa, Touk Saadi, and Sagvan Saleh", An Adaptive Neighborhood Search for k-Clustering Minimum Bi-clique Completion Problems.

[5]"Stefan Ropke", Parallel large neighborhood search - a software framework.

[6]"PaulShaw", Using Constraint Programming and LocalSearch Methods to Solve Vehicle Routing Problems.

END

【如对代码有疑问，可联系小编，可以提供有偿辅导服务】
【有偿辅导纯属个人行为，与团队无关】

赞 赏

长按下方二维码打赏

感 谢 您，
支持学生们的原创热情!

郑重承诺

打赏是对工作的认可
所有打赏所得
都将作为酬劳支付给辛勤工作的学生
指导老师不取一文

精彩文章推荐

干货 | 变邻域搜索算法(VNS)求解TSP（附C++详细代码及注释）

干货 | 变邻域搜索算法(Variable Neighborhood Search,VNS)超详细一看就懂

干货 | 遗传算法(Genetic Algorithm) Java 详细代码及注释

干货 | 遗传算法(Genetic Algorithm) （附代码及注释）

干货|迭代局部搜索算法(Iterated local search)探幽（附C++代码及注释）

干货 | 用模拟退火(SA, Simulated Annealing)算法解决旅行商问题

---The End---

文案 && 编辑：邓发珩

审稿：江玉萍

指导老师： 秦时明岳（华中科技大学管理学院）

如对文中内容有疑问，欢迎交流。PS:部分资料来自网络。

如有需求，可以联系：

秦虎老师（professor.qin@qq.com）

邓发珩 （华中科技大学管理学院本科二年级：2638512393@qq.com、个人公众号：程序猿声）

江玉萍 （南京大学工程管理学院研究生三年级：jiangyuping@smail.nju.edu.cn）

扫一扫，获取数据和模型

还有更多算法学习课件分享哟

文章已于2019-01-02修改

喜欢此内容的人还喜欢

三十而已，我们都变了：我发现她的结局才是人间真实

大大的世界和小小的人儿

人工智能引发的伦理问题及其规制

算法与数学之美

掌握pandas中的时序数据分组运算

Python大数据分析