来沪学术报告之七

I. 计算复杂性理论
II. 目的程序的全局优化

报 告 人：美国国际商业机器公司顾问　施瓦茨及其夫人爱伦
记录整理：上 海 无 线 电 十 三 厂
上 海 计 算 技 术 研 究 所
华 东 计 算 技 术 研 究 所

（内部资料·注意保存）

上海科学技术情报研究所

一九七三年九月

说　　明

美国 IBM 公司顾问、纽约大学数学研究所教授施瓦茨皆其夫人爱伦，于一九七三年六、七月来我国进行友好访问。

据称，施瓦茨是计算机语言研究方面的权威。今年七月初，施瓦茨及其夫人分别在本市作了关于计算机软设备方面的学术报告。

这次学术报告会由上海无线电十三厂夏成林同志主持，会议记录由上海无线电十三厂、上海计算技术研究所及华东计算技术研究所负责整理，但未经报告人审阅，仅供参考。

上海科技情报研究所

计 算 复 杂 性 理 论

大约七年前，才开始进行程序设计理论性方面的工作。以后继续深入发展，这样就逐步为程序设计提供理论基础。这理论的题目叫计算的复杂性。更精确的讲，还有关于算法的问题。两者紧密联系着向前发展。在讨论算法时，人们都想发现好的算法，在计算复杂性领域中，希望计算条件尽可能低些，使算法尽可能实现。要经常分析哪些是做不到的。这两者汇集一起考虑，就能更好理解怎样计算才是合理的。

要与实际对口径，考察一些实际提出的非常困难的问题。这方面的理论已给实际提供了越来越多的启发性的东西，有了一些重大的发现。在一次报告中要对七年的发展做小结是不可能的。但我仍希望能在此报告中向大家介绍一些方法性的和观点性的问题，作些类比。

1. 计算复杂性理论与计算机的关系好比热力学与蒸汽机的关系。

2. 要建立最优性能的绝对的标准。

3. 复杂性理论是要解释计算机科学中的一些重要关系：

a. Munimum size of description.

就是要用最小的规模描述计算过程，即找出最短的程序。从经验知道，有些问题是难以找到这种描述的。

b. Munimum length of computation if performed serially.

c. Munimum size of computation if parallelism possible.

d. Munimum requirements for memory and other computational resource.

4. Interrelations between these invarionts.

例：把程序编长，就能缩小计算规模。

5. 方法：

a. Inter-Reducibility of problems.

即问题的内部简化。

b. Self-reference.

自参考，这个问题可历史地回顾到集合论中 Cantor 的 "diagonal argument".

c. Use of formal models of computation.

其中 b 与 c 是从数理逻辑得来的。

d. Counting and combination arguments.

举例：

例 1：矩阵乘法

这是计算中一个很基本很重要的部分，特别在"大"问题中，应用很广，有成千上万(小时)的时间化在计算矩阵上。

设矩阵是 n 阶

$$a_{ij}=\sum_{k=1}^{n}b_{ik}c_{kj}$$

要求 n^3 次乘法

n^3 次加法

这被称为 "n^3 算法"。

可以提出这样的问题：

是否存在一个小的算法？

一个 $\frac{1}{2}n^3$ 算法？

一个 $c\cdot n^{3-\varepsilon}$ 算法？ (c 为常数)

没有明显的好的方法，但并不是没有方法。五年前开始研究这一理论问题，起初是想证明不存在性。但所有这类企图都失败了。因此就设法找。开始是找

有否 $\frac{1}{2}n^3$ 算法?后来发现，$c\cdot n^{3-\varepsilon}$ 算法是存在的，当 n 很大时，此法比 n^3 的要好得多。

快速矩阵乘法，是德国 STRASSEN 作出的。方法基础是二阶的，本要 8 次，实际只需 7 次乘法。证明是通过代数等式。

A formation of acficient with only 7 multiplication.

$$c_1=(a_{11}+a_{22})(b_{11}+b_{22})$$

$$c_2=(a_{21}+a_{22})b_{11}$$

$$c_3=a_{11}(b_{12}-b_{22})$$

$$c_4=a_{22}(-b_{11}+b_{21})$$

$$c_5=(a_{11}+a_{12})b_{22}$$

$$c_6=(-a_{11}+a_{21})(b_{11}+b_{12})$$

$$c_7=(a_{12}-a_{22})(b_{21}+b_{22})$$

从而积矩阵元素是 c_i 的组合：

$$a_{11}b_{11}+a_{12}b_{21}=c_1+c_4-c_5+c_7$$

$$a_{21}b_{11}+a_{22}b_{21}=c_2+c_4$$

$$a_{11}b_{12}+a_{12}b_{22}=c_3+c_5$$

$$a_{21}b_{12}+a_{22}b_{22}=c_1+c_3-c_2+c_6$$

工作量 $=7$ 次乘法，18 次加法

上述公式虽然只针对二阶，但是非常重要的，小 a 和 b 换成矩阵，等式仍然成立。也即可推广到分块矩阵，

$$\left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] \qquad \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

若记 M_n 为对 n 阶矩阵所需乘法次数，A_N 为相应加法次数，我们有

$$M_{2N}=7M_N$$

$$A_{2N}=18N^2+7A_N$$

因而 $$M_{2^N}=7^N$$

令 $2^N = K$,

则 $M_K = K^{\log 7} \approx K^{2.7}$

矩阵阶数越高，效果越好，

可参考：D. Knuth

The Art of Computer Programming.

该书计划出七卷，已出了三卷，每卷600页左右，每年出一卷。第三卷已带来了。此书很好，我建议你们译成中文。

例2. 寻找 N 个量 $x_1, \cdots, x_N$ 的中位数(Median)问题

当 $x_1, \cdots, x_N$ 是按次序排列时，Median 就是在中间的那个量，即

$$\text{Median} = x_{[N/2]}$$

若不按次序排列时，如何找 Median?

I. 明显(Naive)的办法

先把它们按次序排列，然后找出 Median

Sort quantities. then pick middle element in sequence.

Work of sorting $\sim CN \log N$.

Median must take At least N step, Does it really need $N \cdot \log N$ steps? 这个问题直到去年才解决。

II. 改进了的方法，只须 $C \cdot N$ 步。

① 把 $x_1, \cdots, x_N$ 分成5个子部分。

$T_1 \cdots T_M$ $M = N/5$

$U_1 \cdots U_M$ $T_J \geqslant U_J \geqslant V_J \geqslant W_J \geqslant Y_J$

$V_1 \cdots V_M$ 工作量：$7 \cdot N/5$ 次比较

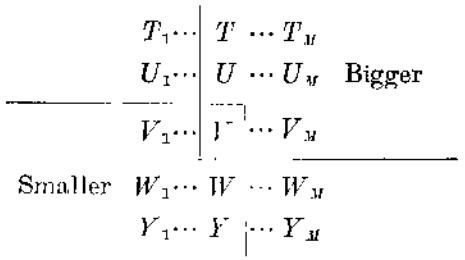
$W_1 \cdots W_M$ 每一列查一查须7次，请大家思考

$Y_1 \cdots Y_M$ 一下为什么。

② 查出 V_i 的 Median V。用刚才一样的方法找，就是递归。

其工作量为 $C_{N/5}$，这里 C_N 是指从 N 个元素中寻找给定位置元素所需的工作量。

注意：V 不在底部 3/10 处(V is not in bottom 3/10)，也不在顶部 3/10处(V is not in top 3/10)。



与左上和右下角的元素间的关系尚不知道。

③ 寻找 V 在所有 x_1, …, x_N 中的精确的位置

工作量：N 次比较

设位置＝第 K 位置

④ if wish element at position $\geqslant K$, throw away elements $<V$, otherwise, throw away elements $>V$.

工作量：实际上无

注意：最多只剩下 $7N/10$ 个元素，即至少有 $7N/10$ 个元素不用再查。

⑤ 在剩下的这些元素中寻找在所要求的位置的元素，其工作量$\leqslant C_{7N/10}$。

总的：

$$C_N \leqslant C_{7/10N} + C_{N/5} + \frac{7}{5}N + N$$

$$\therefore \quad C_N \leqslant K \cdot N$$

这里必须满足

$$K \geqslant \frac{7}{10}K + \frac{1}{5}K + \frac{9}{5}$$

$$\frac{K}{10} \geqslant \frac{9}{5}, \quad K \geqslant 18$$

即 $C_N \leqslant 18 \cdot N$。(若分成 7 个子部分，则更好，有 $C_N \leqslant 15 \cdot N$)

这个问题化了一年时间才解决(在去年解决的)。

N^k— …是幂次般困难(polynomal difficulty)的问题。

是否有指数般困难的问题？去年才首次证明了，在那时已有了相当的研究，

把这些问题分类，是指数般的还是幂次般的困难。在美国流行这样的看法，如果是幂次般的困难还不算困难，高于此类的才算困难。下面再举个例子。

可满足性(satisfiability)

给出逻辑子句

$$x_{i_1} \vee \cdots \vee \bar{x}_{i_r} = c_1$$

$$\bar{x}_{j_1} \vee \cdots \vee x_{j_s} = c_2$$

$$x_{k_1} \vee \cdots \vee \bar{x}_{k_r} = c_M$$

其中 $x_i (i = 1, 2, \cdots, N)$ 是逻辑变量，取值 0 或 1。

问是否存在对每个变量的某种赋值(真和假)使每个子句为真？

这个问题是非常困难的，看来不实用，但与其它实用问题有关。

可满足性问题的显见的算法：

对所有变量试验值真和假的所有组合。

工作量：2^N

是一指数般低劣的算法(Exponentially bod algorithm)。

问题真的这么艰巨(hard)吗？或者是否存在更好的算法？几乎证明了，2^N 确实是困难的问题，证明不长也并不太复杂。

关于问题的内部简化的概念。

Given problem P_1, an amount of work polynomial in size of P_1, will reduces it to P_2.

Then P_1 is 'polynomially reducible' to P_2.

We write

$$P_1 < P_2$$

许多问题，可彼此化来化去，这是用不大严格的话来讲的。它们的难度是一样的，其中一个容易，其它也容易，一个难，其它也难。有些是理论性很强的问题，有些是实用性很强的问题。

Berkely 大学的 K. Karp 和加拿大多伦多大学的S. Cook 所介绍的想法被其他许多人在解决同类问题时应用了。

下面介绍几个问题(注：Schwatz 教授列了一张表，大概有十几个问题，大都可以互化的，因时间关系，他只列举了下面几个)：

1. Satisfiability.

2. 0—1 integer programming.

input：integer matrix c and vector d.

output：(即是否存在)向量 x 使 $cx=d$, x 的元素是 0—1。

3. ……

……

8. Chromatic number：

给出一些图，如图 1。

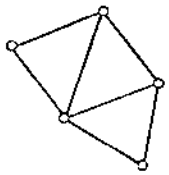


图 1

在每个结点上画上颜色，相邻结点不能有相同颜色，找最小的颜色数。

这个问题粗看是个艺术上的问题，但对编译程序也很有用，把结点看成变量或程序，搞编译的人要把它放进机器。如同时需要，两节点间就用边连结，否则就不连。如不同时需要，可把量放第一个寄存器，则这一问题就变成编译时寄存器的最小数问题。还有很多问题，有些与图论、集论、数理逻辑、实用问题等有关。

9. Knapsack.

给出 S_1, …, S_N; T

问能否选出 S 的子集，使

$$S_{i_1}+\cdots+S_{i_k}=T$$

如果把 S 想象为货物，T 想象为卡车，则这就变成一个实际问题了。

有理由认为，最基本的问题是第一个问题。

座　　谈

问： 计算复杂性理论有哪些定理？数值计算中如何应用？

答： 近来做了许多研究，具体不太清楚，但可举些例子谈谈。

作者有　S. Winograd (IBM)及 M. Rabin，前者对矩阵乘法作了研究(关于

矩阵内积 $\sum_{i=1}^{n} x_i y_i$)，他证明了上午介绍的方法，也只证明了二阶矩阵，三阶矩阵尚未证明；对算术运算、特别是乘法运算与硬件结合起来进行研究，说明硬件乘法没有特别好的技巧。做乘法须$\log N$步，用 Carry-jumping adder (跳跃进位加法器)。上述2人一起研究了牛顿法解代数方程的收敛速度问题，证明了在不同的条件下，最好的方法是没有的。

报告会中所指的书的第二卷中，semi-numerical algoriethms 收集了很多实用性较大的算法，快速富里哀变换，直接的方法

$$\int e^{xy} f(y)\,dy$$

需执行 n^3 项，其中 n 是区间节点个数。改进成 n^2 次即可，也有一些多项式的算法。

问： 上午例一中，乘法的次数减少了，但计算过程长了，复杂性增加了，是否合算？

答： 增加的困难是很少的，主要也取决于程序库的情况。

问： 有人强调程序的简单性，运行时间考虑较少，与上午的说法相比如何？

答： 先作一些其它的说明，矩阵乘法的新方法可引用于线代数的迭代法，效果比高斯消去法更好，你们用的是那种程序库？

问： 用 ALGOL 写程序库，请问美国一般的程序库介几类？

答： 程序库很大，我们研究所用得较多的程序可直接从磁盘读入，例如矩阵乘法，也可从卡片上调入。教授使用计算机因时间有限而往往使用高效率程序。

问： 计算复杂理论与算法关系比较大，与程序关系如何？

答： 算法是程序的一张计划表，好的算法当然亦有好的程序。

问： 请谈谈美国目前软件发展概况。

答： 近年来，美国对语言的兴趣并不太高，五年以前对可扩充语言兴趣较大，去年，兴趣又高涨了，谈一谈历史的看法，当这种方法只取得了部介成果时，大家有一定的失望情绪，认为语言不是很重要的，后来因为国际上知识界的赶时髦，计算技术新东西多，经典的东西未建立，也未标准化；现在有二种趋势：

1. 建立集合论，开始也是赶时髦，现在这并不是主要原因，我本人三年前就已有兴趣了。其他人则是去年才开始感兴趣，这与哈佛大学的基塔姆教授工作有关，美国国家几个研究机构、新墨西哥州研究机构有这种趋势，这工作在 Wells 指导下进行，语言有 MADCAPVI。

2. 人工智能的发展，但应用不多，麻省理工学院、斯坦福大学、其它大学做了一些工作。R. Floyd 做了这方面的工作。这方面的语言有"PLANNER"、"QA4" 及 "CONNIVER"，其他从事该工作的有：

Sussman、Hiwiff、Perksem、Stanford、Fikes，大致介绍此种语言。

关于人工智能，有人要编下棋程序，下棋的人不知下一步如何走才算最好，以通常的方式看这个问题，问题被分成二部机器进行计算，各执行一部分，形成树形结构，直到最后得出正确的决定，Floyd 采用这种思路，在程序语言上实现，发展了一种语言，使人觉得这个问题似乎是解决了。

任何时候，需要作出决定而计算不正确时，整个内存被占用，然后重新回到此问题，并另选一线路继续进行计算，这并不省多少工作。不本的是这里效率很低，因为尚谈不上优化，我认为，今后 10～15 年达不到实际应用。理论上有兴趣者比我乐观，美国有些大学很多人对此感兴趣。

问："BASIC" 语言的结构上，您有何看法？

答：应用不多，据了解美国分时系统较普遍；但只是大中学的教学上用 BASIC；纽约大学用它教学生学会初步的程序设计，学起来比 ALGOL 简单。

问：请谈谈纽约大学计算机科学的教学内容。

答：有如下课程：鼓励学生学许多数学课程，微积分，线代数，抽象代数，概率论，数值分析，程序知识有：FORTRAN 与 FORTRAN 很不相同的 SNOBOL。这是第一学期；汇编语言，数据结构(基本技巧、列表、递归)，LISP 语言，这是第二学期；编译语言分绍，这是第三学期。三年级介绍一些硬设备，用五个月时间编写一些较长的程序，其它尚有外语等文化课，四年期间就是这些内

容、另有选修课，特别重视应用数学。

关于研究生方面，分考博士、硕士之类。后者只要一年多，前者需3～4年的时间；白天上班，晚上到学校学习者，多半系考副博士，上一、二门课程，实用性比较强；全日学习者，理论课程应用性课程都学习，多数后来教书或进入计算机工业工作。计算机科学教授的位子不断在增加。

考硕士学位者，课程有：编译系统，操作系统、硬件、算法语言、计算理论、计算复杂性理论，共需半年。线性规划、概率论、计算方法。教师有专职和兼职教师，后者来自计算机工业。

考试情况：研究生第一年期末笔试，五个题目选三个，包括线代数，基本算法，编一程序，形式语言。最后是口试，然后才写学位论文。

问： 操作系统需大量人力物力，目前有何新的发展？

答： 问题的回答是困难的，人们对其结构的了解很差，许多问题处于模糊状态，这是存在的关键问题。当人们的认识提高时，问题便可以简化。我希望许多学得比较好的学生从事此一工作，用集合论语言进行研究，加上操作系统本身的条件，例如中断，简明扼要地写出来，快写完了，一共只有十二页，了解它的可能性比较大。

从事操作系统的教学是困难的，来有人全面地介绍过，状况不是令人满意的。有一本文献：200多页，去年出版，名叫：Multics（麻省理工学院）。

问： 请谈谈关于语言形式化的看法，何种方法较有前途？对编译系统结构的影响如何？

答： 这方面工作并不是十分重要，也许别人不同意此看法，数学问题经常来自数学实践，有时由于数学家的爱好而把问题的内容丢掉，而象汽球一样离开了地面；当然，它可以部分地帮助我们理解编译系统的句法问题，所以一些研究是有意义的，其它部分无多大意义。“计算复杂理论”比它重要得多，即使如此，计算复杂理论最终有些研究也是无用的，这就是理论联系实际的问题了。

目的程序的全局优化

局部优化只用到分程序中的信息，而全局优化要用到整个程序中的信息。举一个例子，在图 1 中，节点代表基本分程序，线条代表程序走向。

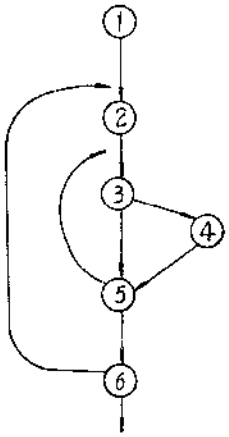


图 1

所谓基本分程序，指的是这样的一连串指令，它只有一个入口和一个出口，分别在头尾处。

第一点，要找出变量的定义（指被赋以值）与其应用之间的关系。定义会影响使用。若定义的量被赋以一个常数，而此变量又在下面（值不变）参加与另一常数进行的运算，则可直接把该变量用前一常数换进去并在编译中算好，这方法称为常数的传递。

第二点是要找出有效区域(Find live area)。

这主要是为了寄存器的分配，如图 1 中，可从 ②→③→④，但亦可有 ⑤→③→④，所以在 ② 或 ⑤ 中定义的量，均可在④中被用到，如这些量都想利用寄存器，就存在一个分配问题。这一般分三步考虑：

1. 找出它们的关系；

2. 使用定义及其应用之间的关系；

3. 选用哪些寄存器。

但寄存器的分配问题是一个组合性的问题，并未完全解决，这方面较好的是 ERSHOV 对 α–ALGOL 所做的工作。

现在回到讨论如何有系统地找出上面所说的关系。

首先要有系统地收集信息。

在每一基本块中，可能出现的量值可分为如下三类：

第一，upwards exposed uses 的，记为 ub_i；

第二，Downwards available definition 的，记为 $\overline{D}b_i$；

第三，Definition not killed，记为 $\overline{K}b_i$。

对每一节点中所有变量，均分析之，收集得所有三种信息，为减少信息的存贮容量，可以二进数位来表示信息的特征。

现考虑一个没有循环的程序，即其中只有向前的转移而没有向后的转移(图 2)。

看如何把一个节点上的信息提供给整个程序。

把在第 i 节点中所得到、并可能影响下面使用的信息记为 A_i。

则
$$A_i = R_i \wedge \overline{K}b_i \vee \overline{D}b_i$$

其中 $R_i = u_p A_p$ (p 指前面的一些节点)，即 A_i 为(上面下来的信息)“与”(在本身中不消失的信息)“或”(本节点内产生的信息)。

〔然后爱伦以一个简单的例子解释了一下这个意思(略)。〕

下面再看一个在程序中有循环的情况(图 3)。

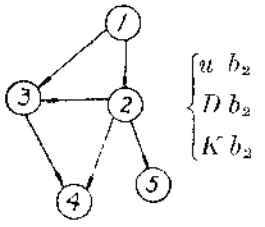


图 2

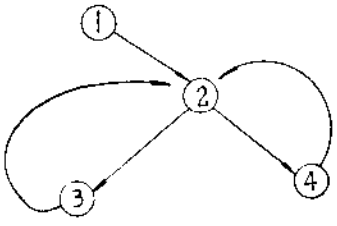


图 3

对这种情况没什么好的处理方法，因为每一处的定义均可影响别处的使用。

让我们引进 interval 的定义，interval 是 a maximal single entry subgraph in which all loops pass through the entry node，对这种 interval 的处理举一例子，见图 4，这是一个 interval，对此，可首先把 ②→① 及 ⑤→① 这二条线去掉，作为无循环的情况处理，然后加入这二条线再处理，便可得最后结果。

图 5 是表示一个有好几个循环的程序的例子。

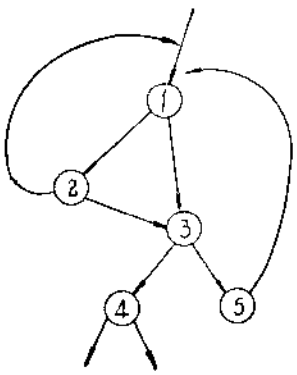


图 4

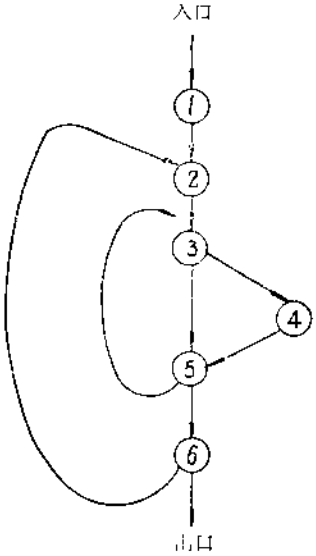


图 5

把它按最大化间隔分成四部分：

间隔 I[1] 包括 ①

间隔 I[2] 包括 ②

间隔 I[3] 包括 ③ ④ ⑤

间隔 I[6] 包括 ⑥

其中，仅 I[3] 这一间隔包含几个基本分程序，对它可用分析图 4 相同的办法进行分析。然后可将图逐步归并。用 ⑦ 代表 I[3] 这一块，便可得图 6；再把 ②；⑦，⑥ 合成一块，便得图 7，最后，再把二块合看成一块，得图 8。

这样推导的意义，在于能够把一个比较大的程序逐步归并。于是，总的分这么二步。第一步，将图从 Lower → Higher，对每一图中的每一节点决定 Db_i，$\bar{K}b_i$，ub_i。第二步，将图从 higher → lower，对每一图的每一节点决定 R_i（进来

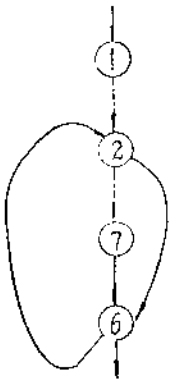


图 6

图 7

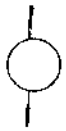


图 8

的量）及 A_i（提供给下面的量），在这步中，要考虑反回线（见图 5 中的 ⑥ → ② 及 ⑤ → ③）。

以上是程序优化的基础内容，其中不作变换，仅仅收集信息，基本上包括下面二部分：

一、控制流的分析

① 在程序本身中找出基本块。

② 构造出图形。

③ 找出间隔，在此基础上归并推导图形。

二、数据流的分析

① 找出在每一基本块中的信息 Db_i, $\overline{Kb}_i$, ub_i。

② 找出定义和使用之间的关系（用上面所讲的控制流程信息和基本块信息 R_i, A_i）。

参考文献有：

Allen "A basis for program optimization" IFIP Proceedings 1971

再提一下最佳化的次序

constant propagation { Control Fow / Data Fow

{ Redundant code elimination and code motion } 只有用到 control flow

changing multiplies to adds

关于这二点，举了一个避免重复计算子表达式的简例(略)。又举了一个简例说明将子表达式向循环外提出的情况(略)。

又谈到前面的方法，对于下面两种情况无法判别出公共子表达式，即

1. 存在两个表达式 $A*B$ 及 $C*B$ 但满足条件 $A=C$ 的情况。

2. 存在两个表达式 $A*B$ 及 $B*A$ 的情况。

座　　谈

一、优化方面

1. 全局优化及交错程序的优化

Cross Program 的优化特别重要，因为这是构成程序的一个方向，且涉及最高级语言的问题。

2. 研究 PL/1 的优化

1) 递归问题。

2) Pointers。

3) Condition Monitering。

对 2)，相对于 ALGOL, Pointers 在 PL/1 中很重要，因为存贮器有重迭，数据存区边界在改变，所以首先要了解指示器。举例：

$A=$

Declare A (10) Float;

Declare B Float Baseo D

$P=$address [$A(2)$]

$P \rightarrow B=0$

↳changes A

这一分析就很困难。

二、现在工作是怎样写编译程序

一个正在试验的系统中写 compiler 的次序