ZIMPL 用户指南

(祖萨研究院数学规划建模设计语言)

托尔斯滕 • 科赫

版本 3.7.0

2024 年 10 月

目录

摘要

ZIMPL 是一种轻量化的特定领域语言 (little language)，用于将问题的数学模型描述转译为线性或 (混合) 整数规划程序，并保存为 (希望是) 能被 LP 或 MIP 的求解器求解的 LP 或 MPS 文件格式。

1 序言

愿源码与你同在，卢克！[1]

许多 ZIMPL 中的功能 (以及更多它不具备的功能) 都可以在罗伯特 • 富勒、大卫 • N • 盖伊和布莱恩 • W • 克宁翰合著的关于 AMPL 建模语言的优秀书籍 [FGK03] 中找到。如果您对当前 (商用) 建模语言的最新进展感兴趣，也可以参考文献 [Kal04b]。

但另一方面，拥有程序的源代码可能带来许多优势。例如，能够在不同的架构和操作系统上运行，能够根据需求对程序进行修改，以及不必与许可证管理器纠缠的便利性，都可能使一个功能弱得多的程序成为更好的选择。正因如此，ZIMPL 应运而生。

迄今为止 ZIMPL 被逐步完善并成熟，已被应用于数个工业项目和高校教育课程，展示出其不仅在应对大规模数学模型时，也在面向学生教育中，具备出众的能力。而这也离不开我的早期用户阿明 • 菲根舒 (Armin Fügenschuh)，马克 • 普费奇 (Marc Pfetsch)，萨沙 • 卢卡茨 (Sascha Lukac)，丹尼尔 • 容格拉斯 (Daniel Junglas)，约尔格 • 兰鲍 (JörgRambau) 和托比亚斯 • 阿赫特贝格 (Tobias Achterberg)，感谢他们提出的意见和问题反馈。特别感谢图奥莫 • 塔库拉 (Tuomo Takkula) 对本手册的修订。

ZIMPL 基于第三版 GNU 宽通用公共许可证发布。更多关于自由软件的信息另请参见 `http://www.gnu.org`。ZIMPL 的最新版本可以在`http://zimpl.zib.de` 找到。如果你发现了任何的程序错误，请发送电子邮件到邮箱`mailto:koch@zib.de`，请不要忘了附上示例来展示问题。如果有人开发了 ZIMPL 的功能扩展，我很乐意收到补丁，并将这些改进纳入主发行版本。

在出版物中引用 ZIMPL 的最佳方式是引用我的博士论文 [Koc04]

```
@PHDTHESIS{Koch2004,
  author      = "Thorsten Koch",
  title       = "Rapid Mathematical Programming",
  school      = "Technische {Universit\"at} Berlin",
  year        = "2004",
  url         = "http://www.zib.de/Publications/abstracts/ZR-04-58/",
  note        = "ZIB-Report 04-58"
}
```

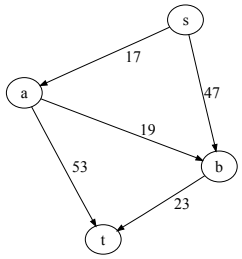
[1] **译者注：**“愿原力与你同在”(May the force be with you.)，是《星球大战》系列影视作品里一句著名台词，最初是影片故事里对拥有原力者的一种祝福和祈祷。这里原文作者利用了谐音，将“原力”(force) 替换为了“源码”(source) 以祝福支持开源事业的读者。

2 简介

考虑一个 $s-t$ 最短路问题的线性规划形式，针对有向图 (V,A)，边的成本系数记为 c_{ij}，对于集合 A 中的所有边 $(i,j)\in A$，建模如下：

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ & \sum_{(iv)\in\delta^-(v)} x_{iv} = \sum_{(vi)\in\delta^+(v)} x_{vi} \quad \text{for all } v\in V\setminus\{s,t\} \\ & x_{ij}\in\{0,1\}, \text{for all i,j in A} \end{aligned} \tag{1}$$

其中，对于任意 $v\in V$，定义 $\delta^+(v) := (v,i)\in A$ 和 $\delta^-(v) := (i,v)\in A$。对于一个特定的图，模型可以具象表述为：



$$\begin{aligned} \min \quad & 17x_{sa}+47x_{sb}+19x_{ab}+53x_{at}+23x_{bt} \\ \text{subject to} \quad & x_{sa}=x_{ab}+x_{at} \\ & x_{sb}+x_{ab}=x_{bt} \\ & x_{ij}\in\{0,1\}, \text{for all i,j} \end{aligned}$$

现将此类问题输入求解器所使用的标准格式称为 MPS，这一格式系 IBM 为 20 世纪 60 年代的数学规划系统 System/360 设计的 [Kal04a, Spi04]。尽管几乎所有现存的线性规划 (LP) 和混合整数规划 (MIP) 求解器都能读取这种格式，虽然 MPS 格式非常适合通过穿孔纸带输入，且对计算机来说也至少是可读的，但对人类而言，却几乎不可理解。例如上述线性规划问题的 MPS 文件内容如下：

```
NAME          shortestpath.lp
ROWS
 N  Obj
 E  c0
 E  c1
 E  c2
COLUMNS
    INTSTART  'MARKER'        'INTORG'
    x0        Obj         17  c0           1
    x0        c2           1
    x1        Obj         47  c1           1
    x1        c2           1
    x2        c1           1  c0          -1
    x2        Obj         19
    x3        Obj         53  c0          -1
    x4        c1          -1  Obj         23
RHS
    RHS       c2           1
BOUNDS
 UP Bound     x0           1
 UP Bound     x1           1
 UP Bound     x2           1
 UP Bound     x3           1
 UP Bound     x4           1
```

```
ENDATA
```

而另一个可能的格式是 LP 格式 [ILO02] 相比之下具有更高的可读性[2]，很接近具象化的表述，但是只被极少数的求解器所支持。

```
Minimize
 Obj: +17 x0 +47 x1 +19 x2 +53 x3 +23 x4
Subject to
 c0: -1 x3 -1 x2 +1 x0 = +0
 c1: -1 x4 +1 x2 +1 x1 = +0
 c2: +1 x1 +1 x0 = +1
Bounds
 0 <= x0 <= 1
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
Generals
 x0 x1 x2 x3 x4
End
```

而鉴于其中又必须精确指定矩阵 A 的每个参数，这仍算不上是一种数学建模语言的理想选择。

抽象公式表述

而现在，这一模型在 ZIMPL 中可以写作：

```
set V      :={"a","b","s","t"};
set A      :={<"s","a">, <"s","b">, <"a","b">, <"a","t">, <"b","t">};
param c[A] := <"s","a"> 17, <"s","b"> 47, <"a","b"> 19, <"a","t"> 53,
              <"b","t"> 23;
defset dminus(v) := {<i,v> in A};
defset dplus(v)  := {<v,j> in A};
var x[A] binary;
minimize cost: sum<i,j> in A: c[i,j] * x[i,j];
subto fc:
      forall <v> in V - {"s","t"}:
        sum<i,v> in dminus(v): x[i,v] == sum<v,i> in dplus(v): x[v,i];
subto uf:
      sum<s,i> in dplus("s"): x[s,i] == 1;
```

——请将这段代码与 (1) 相比较。将这段代码输入 ZIMPL 即可自动生成 MPS 或 LP 文件。

像 ZIMPL 这样的建模语言的价值在于它们具备直接处理数学模型本身的能力，而不是仅仅对系数进行处理。除此之外，模型的具象 (可以理解为是生成的 LP 文件或 MPS 文件) 通常会通过外部数据生成。从某种意义上来讲，“具象化”正是将模型应用于外部数据的结果。在我们上面的算例中，所谓外部数据

[2] LP 格式也具有一些乖张的限制。比如变量不能被命名为 e12 或者类似的名称，而且也不能指定范围约束。

就是带有成本系数和指定的 s 和 t 的图，而模型则是 st 最短路优化问题的数学公式。当然，ZIMPL 也支持通过文件来初始化模型。例如，上述这个 ZIMPL 脚本的第一行也可以写作：

```
set       V:= {read "nodes.txt" as "<1s>"};
set       A:= {read "arcs.txt"  as "<1s,2s>"};
param c[A] := read "arcs.txt"  as "<1s,2s>3n";
```

而 ZIMPL 也可以根据“nodes.txt”和“arcs.txt”两个文件中的定义生成任何一个最短路问题的实例。这些文件中具体的格式规定将在4.4节中叙述。

3 命令行调用

要对文件 `ex1.zpl` 中给定的模型运行 ZIMPL 程序，需要键入如下命令：

```
zimpl ex1.zpl
```

命令的一般形式为：

```
zimpl [options] <input-files>
```

命令可以接受多个文件输入，按顺序读取，如同被合并为一个单一的大文件。如果在处理过程中产生任何报错，ZIMPL 会打印错误信息并中止运行。若一切正常，结果将根据指定的选项被写入至少两个文件中。

第一个文件是根据模型生成的 CPLEX 、LP 格式，MPS 格式或“人可读”格式的优化问题文件，后缀名分别是 *.lp*，*.mps*，或者*.hum*。而另一个是 *table* (表格) 文件，以后缀名*.tbl* 结尾。表格文件列出了模型中使用到的所有变量及约束的名称，以及它们在优化问题文件中对应的名称。之所以会造成这一名称转译的原因在于 MPS 文件格式中的名称长度限制为 8 个字符。而且在 LP 文件中也同样限制名称的长度。具体限制取决于所使用的版本。CPLEX 0.7 中的限制为 16 字符，且会无视名称中其余的部分，而 CPLEX 0.9 则上限为 255 字符，但部分命令的输出中只会展示前 20 个字符。

ZIMPL 可解析的完整参数列表详见表1。一个典型的 ZIMPL 调用命令如下例所示：

```
zimpl -o solveme -t mps data.zpl model.zpl
```

这将读取文件 `data.zpl` 和 `model.zpl` 作为输入，并生成输出文件 `solveme.mps` 和 `solveme.tbl` 。需要注意的是，如果指定输出为 MPS 格式且优化目标为极大化，目标函数的正负号将被反转。这是因为 MPS 文件格式不支持直接指定目标函数的优化方向，其默认假设为极小化。

4 语法规范

每个 ZPL 文件包含六种类型的声明语句：

- 集合 Sets
- 参数 Parameters
- 变量 Variables
- 目标 Objective
- 约束 Constraints
- 函数定义 Function definitions

每一句语句都以一个分号结尾。除了字符串以外，“`#` ”符号之后到这行末尾的所有内容都会被视作注释忽略。如果一行以单词 `include` 开头，后跟有带双引号的文件名，则会读取并处理该文件，而不是该行。

-t *format*	指定输出的格式。可以是默认的 `lp` 格式，或 `mps` 格式，或者仅供人类阅读的 `hum` 格式。还可以是 `rlp` 格式，此格式与 `lp` 相同，但基于 *seed* 参数提供的随机种子，对行和列进行了随机置换。另一种可能的输出格式是 `pip` 格式，用于描述多项式整数规划 (Polynomial IP) 问题；此外还有 `q` *x* 格式，用于描述二次无约束 0-1 优化 (QUBO，即 Quadratic Unconstrained Binary Optimization) 问题，其中 *x* 为格式选项：*0* 表示使用从零开始的矩阵索引 (默认是从 1 开始)，*c* 表示在文件中使用字符 `c` 作为注释行指示符 (默认是 `#`)，*p* 表示在实例文件的首行写入字符 `p` 。
-o *name*	选择输出文件的文件名不含扩展名。 默认为输入的第一个文件的文件名，不含路径和扩展名。
-F *filter*	将输出通过管道传递给一个过滤器。字符串中的%s 将被替换为输出文件的文件名。举个例子：参数 `-F "gzip -c >%s.gz"` 可以压缩所有输出的文件。
-l *length*	设置 `lp` 文件格式中变量和约束的最大长度到 *length*。
-n *cform*	选择生成约束名称的格式。如果设置为 `cm` ，则约束将以字符 'c' 开头，并被编号为 1...n。设置为 `cn` 时，约束名称将使用 `subto` 语句中指定的名称，并在该语句内部编号为 1...n。设置为 `cf` 时，约束名称将以 `subto` 中指定的名称开头，随后加上编号 1...n (类似于 `cm`)，并附加来自 `forall` 语句的所有局部变量。
-P *filter*	将输入通过管道传递给一个过滤器。字符串中的%s 将被替换为输入文件的文件名。举个例子：参数 `-P "cpp -DWITH_C1 %s"` 可将输入的文件传递给 C 语言的预处理器对输入文件进行处理
-s *seed*	用于随机数生成器的一个正值的随机种子 `seed` 。例如 `-s 'date +%N'` 可提供不断变更的随机种子。
-v *0..5*	设置输出日志的详细等级。0 表述静默，1 为默认水平。2 为详细输出，3 和 4 为细致输出，5 为调试级信息。
-D *name=val*	设置参数 *name* 为指定值。这相当于在代码开头增加了 `param name:=val` 一行内容。如果 ZIMPL 文件中已经声明了同名的参数而 `-D` 选项又同样设置了相同的名称，则以`-D` 的指定为准。
-b	启用 bison 语法解析器的输出。
-f	启用 flex 词法分析器的输出。
-h	显示帮助信息。
-m	生成一份 CPLEX `mst` (Mip STart) 文件
-O	尝试通过预处理来简化生成的线性规划模型。
-r	生成一份 CPLEX `ord` 分支次序文件。
-V	显示版本号。

表 1: ZIMPL 参数选项

4.1 表达式

ZIMPL 基于两种最基本的数据类型：字符串和数值。凡是需要提供数字或字符串的地方，也可以使用对应值类型的参数。在大多数情况下，可以使用表达式作为值，而不仅仅是写一个数字或字符串。运算优先级通常取决于一般规定，但可以使用括号来显式指定求值顺序。

数值表达式

ZIMPL 中的数字可以通过一般的写法给定，如 2，-6.5 或 5.23e-12。数值表达式的形式包括数字、具有数值类型值的参数，以及表2列出的任何一种运算符或函数。除此之外表3中所示的函数也是可以使用的。需要注意的是，这些函数仅使用普通的双精度浮点运算进行计算，因此精度有限。关于如何使用 `max` 和 `min` 函数的例子，可以在第11页的第4.3节中找到 [3]。

`a^b`, `a**b`	a 的 b 次方	a^b，b 必须为整数
`a+b`	加法	$a+b$
`a-b`	减法	$a-b$
`a*b`	乘法	$a \cdot b$
`a/b`	除法	a/b
`a mod b`	取模	$a \mod b$
`abs(a)`	绝对值	$\lvert a \rvert$
`sgn(a)`	符号函数	$x > 0 \Rightarrow 1, x < 0 \Rightarrow -1$, 否则为0
`floor(a)`	向下取整	$\lfloor a \rfloor$
`ceil(a)`	向上取整	$\lceil a \rceil$
`round(a)`	四舍五入	$\lfloor a \rceil$
`a!`	阶乘	$a!$，a 必须为非负整数
`min(S)`	集合元素的最小值	$\min_{s \in S}$
`min <s> in S: e(s)`	函数在集合上取得的最小值	$\min_{s \in S} e(s)$
`max(S)`	集合元素的最大值	$\max_{s \in S}$
`max <s> in S: e(s)`	函数在集合上取得的最大值	$\max_{s \in S} e(s)$
`min(a,b,c,...,n)`	列表元素的最小值	$\min(a, b, c, \ldots, n)$
`max(a,b,c,...,n)`	列表元素的最大值	$\max(a, b, c, \ldots, n)$
`sum(s in S) e(s)`	集合元素代入函数计算后求和	$\sum_{s \in S} e(s)$
`prod(s in S) e(s)`	集合元素代入函数计算后乘积	$\prod_{s \in S} e(s)$
`card(S)`	集合的技术	$\lvert S \rvert$
`random(m,n)`	伪随机数	$\in [m, n]$, rational
`ord(A,n,c)`	序数	集合 A 中第 n 个元素的第 c 个分量
`length(s)`	字符串的长度	字符串 s 的字符个数
`if a then b else c end`	条件判断	$\begin{cases} b, & \text{if } a = \text{true} \\ c, & \text{if } a = \text{false} \end{cases}$

表 2: 有理算术函数

[3] **译者注**：经译者实测，表2所列表达式除加减乘除等运算符外，在 ZIMPL 中使用似乎均会报错，正确用法形如 `sum <i> in I: e(i)` 在4.3节中展示。其余函数亦同。为考证这一问题，译者已向 ZIMPL 的 Git 仓库提交了 Issue，详情参考 `https://github.com/scipopt/zimpl/issues/2`。该 Issue 于 1 月 2 日得到了开发者的回复，其中 `max` 和 `min` 函数的形式已经得到了修复，但 `sum` 和 `prod` 仍然存在问题。

`sqrt(a)`	平方根	$\sqrt{a}$
`log(a)`	以 10 为底的对数	$\log_{10} a$
`ln(a)`	自然对数	$\ln a$
`exp(a)`	指数函数	e^a

表 3: 双精度函数

字符串表达式

字符串由双引号“ " ”包裹，形如`"Hello Keiken"` 。两个字符串可以通过“`+` ”运算符拼接，例如，如果使用`"Hello " + "Keiken"` 将会得到`"Hello Keiken"` 。函数 `substr(string, begin, length)` 可用于提取字符串中的特定部分。其中，`begin` 是要使用的第一个字符，计数按照第一个字符从 0 开始。要提取的字符串长度可通过 `length` 函数来确定。

逻辑表达式

返回值为 *true* 或 *false* 的表达式。对于数值和字符串，定义了关系操作符如 <，<=，==，!= 和 >=。逻辑表达式通过 `and` ，`or` 和 `xor`[4]连接，并可通过 `not` 取反。表达式 *元组* `in` *集合表达式* (将在下一章节中解释) 可用于测试元组中集合成员的关系。逻辑表达式也可以在 *if* 语句的 *then* 或者 *else* 的部分中使用。

变量表达式

以下的内容可能是一个数值，字符串或逻辑的表达式，取决于 *expression* 的部分是字符串，逻辑还是数值表达式：

`if` *逻辑表达式* `then` *表达式* `else` *表达式* `end`

同样地，`ord(`*集合*`,` *元组编号*`,` *分量编号*`)` 函数的返回值是集合中某个具体的元素 (关于集合的更多细节将在后文说明)。函数的返回值类型取决于集合中分量 (component) 的类型。如果集合中的分量是数值，函数返回数值；如果是字符串或布尔值，则返回对应类型的值。

4.2 元组与集合

元组是具有固定维度的有序矢量，分量为数值或字符串类型。集合内可包含 (有限多个) 元组。每个元组在集合中都是不重复的。在 ZIMPL 中所有集合都是内部有序的，但没有特定的顺序。集合通过花括号来包裹，形如“`{` ”和“`}` ”。一个集合中的所有元组必须具有相同个数的分量。对于一个特定集合中的所有元组，它们各自的第 n 个分量的数据类型必须相同，也就是说它们必须全都是数值或者全都是字符串。元组的定义通过尖括号 < 和 > 包裹，示例如`<1,2,"x">` 。各个分量通过逗号分隔。如果元组是一维的，则可以在一列元素中省略元组的包裹符，但在这种情况下，定义中的所有元组都必须省略它们。比如 `{1,2,3}` 是合法的定义，而 `{1,2,<3>}` 是不合法的。

集合可通过集合语句来定义，包括关键字 `set` ，集合名称及赋值操作符 `:=` ，以及一个合法的集合表述。

集合通过使用模板元组 (template tuple) 来引用，模板元组由占位符 (placeholders) 组成。这些占位符会被相应元组中各分量的值所替代。例如，一个由二维元组组成的集合 `S` 可以通过 `<a,b> in S` 来引用。如果模板元组中的某些占位符被赋予了实际值，那么只有那些与这些值匹配的元组会被选取 (extracted) 。例如，`<1,b> in S` 只会选取第一个分量为“`1` ”的元组。需要注意的是，如果占位符的名称与已定义

[4] a xor $b := a \wedge \neg b \vee \neg a \wedge b$

的参数、集合或变量的名称相同，那么这些名称会被替换为相应的值。这可能会导致错误，或者被解释为实际的值。

示例

```
set A := { 1, 2, 3 };
set B := { "hi", "ha", "ho" };
set C := { <1,2,"x">, <6,5,"y">, <787,12.6,"oh"> };
```

对于集合表达式，表4中所示的函数和运算符已被定义。

关于形如 *逻辑表达式* `then` *集合表达式* `else` *集合表达式* `end` 的语句形式如何使用的示例可以和下标集合的示例一同在第9页找到。

示例

```
set D := A cross B;
set E := { 6 to 9 } union A without { 2, 3 };
set F := { 1 to 9 } * { 10 to 19 } * { "A", "B" };
set G := proj(F, <3,1>);
# 将会得到: { <"A",1>, <"A",2"> ... <"B",9> }
```

条件集合

集合可以通过一个逻辑表达式表达式加以限定从而取得满足条件的元组。对于通过 `with` 子句给定的表达式，会对集合中的每个元组进行求值。只有当表达式的结果为 *true* 时，相关元组才会被包含在新的集合中。

示例

```
set F := { <i,j> in Q with i > j and i < 5 };
set A := { "a", "b", "c" };
set B := { 1, 2, 3 };
set V := { <a,2> in A*B with a == "a" or a == "b" };
# 将会得到: { <"a",2>, <"b",2> }
set W := argmin(3) <i,j> in B*B : i+j;
# 将会得到: { <1,1>, <1,2>, <2,1> }
```

索引集合

可以使用一个集合对另一个集合进行索引，从而得到一个“集合的集合”。索引集合的访问方式是在集合名后加上方括号“`[`”和“`]`”，例如 `S[7]` 。表 5 列出了可用的函数。对索引集合的赋值有三种方式：

- 赋值表达式可以是一个由逗号分隔的键值对列表，每对元素由索引集合中的一个元组和要赋值的集合表达式组成。
- 如果索引中包含一个索引元组，例如 `<i> in I`，则赋值操作会对索引元组的每个取值分别求解。
- 通过返回索引集合的函数进行赋值。

`A*B`, `A cross B`	叉积	$\{(x, y) \mid x \in A \wedge y \in B\}$
`A+B`, `A union B`	并集	$\{x \mid x \in A \vee x \in B\}$
`union <i>` ` in I: S`	同上，对索引集合取并集	$\bigcup_{i \in I} S_i$
`A inter B`	交集	$\{x \mid x \in A \wedge x \in B\}$
`inter <i>` ` in I: S`	同上，对索引集合取交集	$\bigcap_{i \in I} S_i$
`A\B`, `A-B`, `A without B`	差集	$\{x \mid x \in A \wedge x \notin B\}$
`A symdiff B`	对称差	$\{x \mid (x \in A \wedge x \notin B) \vee (x \in B \wedge x \notin A)\}$
`{n..m by s}`,	生成集合， (默认 $s = 1$)	$\{x \mid x = \min(n, m) + i\lvert s\rvert \leqslant \max(n, m),$ $i \in \mathbb{N}_0, x, n, m, s \in \mathbb{Z}\}$
`{n to m by s}`	生成集合	$\{x \mid x = n + is \leqslant m, i \in \mathbb{N}_0, x, n, m, s \in \mathbb{Z}\}$
`proj(A, t)`	投影 $t = (e_1, \ldots, e_n)$	新的集合将由 n 元组组成，其中第 i 个分量是集合 A 中的第 e_i 个分量。
`argmin <i>` ` in I : e(i)`	极小值	$\operatorname{argmin}_{i \in I} e(i)$
`argmin(n) <i>` ` in I : e(i)`	取 n 个极小值	新集合将由满足使 $e(i)$ 最小的 n 个元素 i 组成。结果可能存在歧义 (由于不同元素可能取得相同的 $e(i)$ 值)。
`argmax <i>` ` in I : e(i)`	极大值	$\operatorname{argmax}_{i \in I} e(i)$
`argmin(n) <i>` ` in I : e(i)`	取 n 个极大值	新集合将由满足使 $e(i)$ 最大的 n 个元素 i 组成。结果可能存在歧义 (由于不同元素可能取得相同的 $e(i)$ 值)。
`if a then b` `else c end`	条件判断	$\begin{cases} b, & \text{if } a = \text{true} \\ c, & \text{if } a = \text{false} \end{cases}$
`permutate(A)`	排列元素	生成一个包含集合 A 中所有元素的排列的元组

表 4: 集合关系函数

示例

```
set I           := { 1..3 };
set A[I]        := <1> {"a","b"}, <2> {"c","e"}, <3> {"f"};
set B[<i> in I] := { 3 * i };
set P[]         := powerset(I);
set J           := indexset(P);
set S[]         := subsets(I, 2);
set T[]         := subsets(I, 1, 2);
set K[<i> in I] := if i mod 2 == 0 then { i } else { -i } end;
set U           := union <i> in I : A[i];
set IN          := inter <j> in J : P[j]; # empty!
```

`powerset(A)`	生成集合 A 的所有子集	$\{X \mid X \subseteq A\}$
`subsets(A,n)`	生成集合 A 包含 n 个元素的子集	$\{X \mid X \subseteq A \wedge \lvert X\rvert = n\}$
`subsets(A,n,m)`	生成集合 A 的包含 n 到 m 个元素的子集	$\{X \mid X \subseteq A \wedge n \leqslant \lvert X\rvert \leqslant m\}$
`indexset(A)`	生成集合 A 的索引集	$\{1 \ldots \lvert A\rvert\}$

表 5: 索引集函数

4.3 参数

参数 (Parameter) 是 ZIMPL 中定义常数的一种方式。参数既可以带有索引集合，也可以不带索引。没有索引的参数只是一个单独的值，可以是数字或字符串；带索引的参数则为索引集合中的每个元素指定一个对应的值。此外，还可以为参数声明一个 *default* 值。

参数的声明方式如下所示：使用关键字 `param`，紧跟参数名，并可以选择是否在方括号中给出索引集合。接着在赋值符号之后，给出一个键值对列表。每对元素的第一个分量是来自索引集合的元组，第二个分量则是该索引对应的参数值。如果只给出一个单独的值，则该值会被赋给参数的所有索引成员。

示例

```
set A := { 12 .. 30 };
set C := { <1,2,"x">, <6,5,"y">, <3,7,"z"> };
param q := 5;
param r[C] := 7;         # 所有元素都是 7
param r[C] := default 7; # 和上一行一样
param u[A] := <13> 17, <17> 29, <23> 14 default 99;
param str[A] := <13> "hallo", <17> "tach" default "moin";
param amin := min A;               # = 12
param umin := min <a> in A : u[a]; # = 14
param mmax := max <i> in { 1 .. 10 } : i mod 5;
param w[C] := <1,2,"x"> 1/2, <6,5,"y"> 2/3;
param x[<i> in { 1 .. 8 } with i mod 2 == 0] := 3 * i;
```

赋值并不需要是完整的。在这个例子当中，并没有给出参数“`w`”下的索引 `<3,7,"z">` 对应的数值。只要该索引值在模型中从未被引用，这种做法就是允许的。

参数表

可以通过表格的方式初始化多维索引参数。这在处理二维参数时尤其有用。数据应组织为表格结构，表格边界由“|”符号标示。随后需提供一行作为表头，表头包含列索引；此外，每一行也必须对应一个行索引。列索引必须是一维的，而行索引则可以是多维的。每个条目的完整索引由“行索引”与“列索引”拼接而成。表中的各项以逗号分隔，并且允许使用任意合法表达式。如下面的第三个示例所示，在表格之后还可以追加一个由条目组成的列表。

示例

```
set I := { 1 .. 10 };
set J := { "a", "b", "c", "x", "y", "z" };

param h[I*J] :=   | "a", "c", "x", "z"   |
                |1|  12,  17, 99,     23 |
                |3|   4,   3,-17, 66*5.5 |
                |5| 2/3, -.4,  3, abs(-4)|
                |9|   1,   2,  0,      3 | default -99;

param g[I*I*I] :=     | 1, 2, 3 |
                  |1,3| 0, 0, 1 |
                  |2,1| 1, 0, 1 |;

param k[I*I] :=  |  7,  8,  9 |
               |4| 89, 67, 55 |
               |5| 12, 13, 14 |, <1,2> 17, <3,4> 99;
```

最后的这个示例等同于：

```
param k[I*I] := <4,7> 89, <4,8> 67, <4,9> 55, <5,7> 12,
                <5,8> 13, <5,9> 14, <1,2> 17, <3,4> 99;
```

4.4 从文件读取集合与参数

可以从文件中读取集合或者参数。其语法是：

read 文件名 as 模板名 [skip n] [use n] [match s] [comment s]

文件名是指要读取的文件名称。模板名是一个用于生成元组的模板字符串的名称。来自每一行输入会被拆分为若干个字段。字段的拆分遵循以下规则：行首与行尾的空格与制表符会被去除；每当遇到空格、制表符、逗号、分号或冒号时，都会开始一个新字段；被双引号包围的文本不会被拆分，并且双引号会被自动移除；若某个字段处发生拆分，则拆分点两侧的空格与制表符也会被移除；若拆分是由逗号、分号或冒号引起的，那么每出现一次该符号就会新建一个字段。

示例

如下的每一行输入都包含三个字段：

```
Hallo;12;3
Moin   7  2
```

```
"Hallo, Peter"; "Nice to meet you" 77
,,2
```

对于元组的每一个分量，都需要指定使用哪一个字段，并在其后标明该字段的类型：若字段应被解释为数值，则写 `n`；若应作为字符串处理，则写 `s`。在模板之后，可以添加一些可选修饰项，顺序不限：`match` s 会正则表达式 s 应用于读取的每一行文本。只有当该表达式与该行匹配时，该行才会被进一步处理。所使用的正则语法为 POSIX 扩展的正则表达式语法。`comment` s 指定一个字符列表，这些字符用于标识文件中的注释。若在一行中遇到任一注释字符，则该行在该字符处被截断。`skip` n 表示跳过文件的前 n 行内容。`use` n 限制读取的行数不超过 n 行。在读取文件的过程中，空行、注释行以及未匹配的行都将被跳过，且不计入 `use` 和 `skip` 的计数范围内。

示例

```
    set P := { read "nodes.txt" as "<1s>" };
nodes.txt:
   Hamburg              → <"Hamburg">
   München              → <"München">
   Berlin               → <"Berlin">

set Q := { read "blabla.txt" as "<1s,5n,2n>" skip 1 use 2 };
blabla.txt:
   Name;Nr;X;Y;No       → 跳过
   Hamburg;12;x;y;7     → <"Hamburg",7,12>
   Bremen;4;x;y;5       → <"Bremen",5,4>
   Berlin;2;x;y;8       → skip

param cost[P] := read "cost.txt" as "<1s> 2n" comment "#";
cost.txt:
   # 名称 价格          → 跳过
   Hamburg 1000         → <"Hamburg"> 1000
   München 1200         → <"München"> 1200
   Berlin  1400         → <"Berlin"> 1400

param cost[Q] := read "haha.txt" as "<3s,1n,2n> 4s";
haha.txt:
      1:2:ab:con1       → <"ab",1,2> "con1"
      2:3:bc:con2       → <"bc",2,3> "con2"
      4:5:de:con3       → <"de",4,5> "con3"
```

与表格式输入一样，可以在 read 语句之后添加由元组或参数项组成的列表。

示例

```
set A := { read "test.txt" as "<2n>", <5>, <6> };
param winniepoh[X] :=
   read "values.txt" as "<1n,2n> 3n", <1,2> 17, <3,4> 29;
```

也可以将一个单独的值读取到某个参数中。在这种情况下，文件应只包含一行内容，或者应通过添加 use 1 参数来指示读取语句仅读取一行。

示例

```
# 读取第五行的第四个值
param n := read "huhu.dat" as "4n" skip 4 use 1;
```

如需读取一个文件中所有的值到元组中，对于字符串值，可以使用 "<s+>" 的模板形式，对于数值，可以使用 "<n+>" 的模板形式。需要注意的是，目前只支持每行最多 65536 个值。

示例

```
# 读取文件里的所有值到一个集合中
set X := { read "stream.txt" as "<n+>" };

stream.txt:
1 2 3 7 9 5 6 23
63 37 88
4
87 27

# X := { 1, 2, 3, 7, 9, 5, 6, 23, 63, 37, 88, 4, 87, 27 };
```

4.5 *sum* 表达式

求和表达式规定为如下所示的形式：

sum *索引* do *语句*

尽管求和表达式可以多级嵌套，但是简单地合并索引可能会更方便。< *索引* > 的一般形式形如：

元组 in *集合* with *逻辑表达式*

可以用一个冒号“ : ”来代替 do ，也可以用竖划线“ | ”来代替 with 。组成“*元组*”的分量的个数必须与组成“*集合*”的元素个数相匹配，with 和“*逻辑表达式*”的这部分是可选的。“*集合*”这部分可以使用任何表达式的形式表述的集合。示例将会在下一节中给出。

4.6 *forall* 表达式

遍历表达式的一般形式为：

forall *索引* do *语句*

可以嵌套多个遍历表达式[5]。*index* 的一般形式和 *sum* 表达式中是相同的。示例将在下一节中给出。

警告！关于 *forall* 和 *sum* 语句中的索引变量作用域

若索引变量已经在 *forall* 和 *sum* 语句外部定义过，则会被替换为其当前已有值。这对于函数定义中的变量也同样成立，如果它们在外部已经定义过，也会发生相同的替换行为。

[5]**译者注**: LaTeX 原文在此处有一行被注释掉的文本，未被显示在 PDF 中。其内容是：“需要注意的是，*sum* 表达式本身也是这里的一个 *语句*，因此可以嵌套或连接它们。”原作者似乎本想提醒读者 *forall* 表达式和前文的 *sum* 表达式可以相互嵌套，但是不知道为什么将该内容隐藏了。

示例

```
k = 5; sum <i,k> in {1..10}*{1..50}: i; # == 55, 因为 k 为定值
forall <k> in K: sum <k,b> in K*{1..3}: z[k,b]; # 等于 |K| 乘以三个元素之和
```

4.7 函数定义

在 ZIMPL 中可以定义函数。函数的返回值必须是数值、字符串、布尔值或集合之一。函数的参数仅能为数值或字符串，但在函数体内部可以访问所有已声明的集合与参数。

函数定义必须以 `defnumb` 、`defstrg` 、`defbool` 或 `defset` 开头，具体使用哪一个取决于函数的返回值类型。其后为函数名及参数名列表，用括号括起来。接着是赋值运算符 `:=` ，后面需给出一个合法的表达式或集合表达式。

示例

```
defnumb dist(a,b)     := sqrt(a*a + b*b);
defstrg huehott(a)    := if a < 0 then "hue" else "hott" end;
defbool wirklich(a,b) := a < b and a >= 10 or b < 5;
defset  bigger(i)     := { <j> in K with j > i };
K = 5; defnum(a) := sum <i,k> in {1..3}*{1..5}: a; # == 3*a
```

4.8 *do print* 和 *do check* 命令

`do` 命令比较特殊，它有两种可用的形式：`print` 和 `check` 。`print` 会将其后所跟的数值、字符串、布尔值、集合表达式或元组输出到标准输出流。这通常用于检查集合是否包含预期的元素，或某些计算是否产生了预期的结果。而 `check` 则必须跟在一个布尔表达式前，若该表达式的值不为 *true*，程序就会中止，并输出一条相应的错误信息。这可以用来断言某些条件是否成立。可以在 `print` 或 `check` 语句之前使用 `forall` 子句。对于字符串或数值类型的值，也可以在 `print` 中给出逗号分隔的多个表达式。

示例

```
set I := { 1..10 };
do print I;
do print "Cardinality of I:", card(I);
do forall <i> in I with i > 5 do print sqrt(i);
do forall <p> in P do check sum <p,i> in PI : 1 >= 1;
```

5 模型

在本节中，我们将运用此前构建的各类机制来表述数学规划模型。除了用于声明变量、目标函数与约束的一般语法外，还提供了一些特殊的语法，用于简洁地表述特殊约束，例如特殊有序集 (special ordered sets) 、条件约束 (conditional constraints) 以及扩展函数 (extended functions) 。

5.1 规划变量

与参数类似，变量也可以带有索引。变量必须属于以下三种类型之一：实型 (称为 `real`)、逻辑型 (`binary`) 或整型 (`integer`)。默认类型为实型。变量可以设置下界与上界。默认下界为零，上界为无穷大。逻辑变量的取值范围始终在零与一之间。可以根据变量的索引值来计算其上下界 (参见示例中的最后一

条语句)。界限也可以显式设置为 infinity 或-infinity 。逻辑与整数变量可以声明为 implicit ，这也就是说，即使变量被声明为连续型，在整数规划的最优解中也可以被假定为取整值。*implicit* 的使用仅在配合支持该特性的求解器时才有意义。截至目前，仅有链接了 ZIMPL 的 SCIP (http://scipopt.org) 支持此特性。在其他情况下，该变量仅会被视为连续变量。可使用 startval 关键字为整数变量指定初始值。此外，可以使用 priority 关键字指定分支优先级。目前，当使用-r 命令行选项时，这些值会被写入一个 CPLEX 格式的 ord 分支顺序文件中。

示例

```
var x1;
var x2 binary;
var x3 integer >= -infinity      # 自由变量
var y[A] real >= 2 <= 18;
var z[<a,b> in C] integer
      >= a * 10 <= if b <= 3 then p[b] else infinity end;
var w implicit binary;
var t[k in K] integer >= 1 <= 3 * k startval 2 * k priority 50;
```

5.2 目标函数

在模型的声明中，最多只能声明一个目标函数。目标函数可以是 maximize (极大化) 或者 minimize (极小化) 的。紧随关键字声明值后的是一个 (目标函数的) 名字，以及一个冒号“：”随后是一个目标函数形式的一个线性表达式。

如果规划目标中包含补偿值 (offset),也就是说一个常数值,ZIMPL 会自动生成一个内部变量 @@ObjOffset 并将其赋值为 1。该变量将以合适的系数被纳入目标函数中。[6]

示例

```
minimize cost: 12 * x1 -4.4 * x2 + 5
   + sum <a> in A : u[a] * y[a]
   + sum <a,b,c> in C with a in E and b > 3 : -a/2 * z[a,b,c];
maximize profit: sum <i> in I : c[i] * x[i];
```

5.3 模型约束

模型约束的一般格式如下所示：

subto [名称]: [项式] [比较符] [项式]

或者，也可以通过如下的形式带区间的[7] (ranged) 约束。

subto [名称]: [表达式] [比较符] [项式] [比较符] [表达式]

[6]这是因为，无论是 LP 还是 MPS 格式，都没有通用的方法在目标函数中直接表示一个常数项。**译者注**: 我不知道原文作者此处所说的“offset”是什么意思，姑且将其翻译为“补偿值”。除此之外，笔者也并未发现 LP 文件不支持目标函数里包含常数项的语法形式。这可能是为了兼容一些比较旧的求解器机制而作的设计。

[7]**译者注**: 原作者在这里指的是类似于 $a \leqslant x \leqslant b$ 这样的约束形式。也就是说，在 ZIMPL 中，形如此的约束不需要分开写成两个式子。

[名称] 表示任意以字母开头的合法名称；[项式] 的定义方式与目标函数中的相同；[比较符] 表示约束的比较运算符，可以是 <= 、>= 或 == 。对于带区间的约束来说，这两个比较符必须相同，且不能为 == ；[表达式] 是任意能够计算出数值的合法表达式。

除了目标函数中使用的线性约束之外，约束语句中还可以包含更高阶 (非线性) 的项。

借助 forall 语句，还可以通过一条语句同时生成多个约束，示例如下所示。

示例

```
subto time:  3 * x1 + 4 * x2 <= 7;
subto space: 50 >= sum <a> in A: 2 * u[a] * y[a] >= 5;
subto weird: forall <a> in A: sum <a,b,c> in C: z[a,b,c]==55;
subto c21: 6*(sum <i> in A: x[i] + sum <j> in B : y[j]) >= 2;
subto c40: x[1] == a[1] + 2 * sum <i> in A do 2*a[i]*x[i]*3+4;
subto frown: forall <i,j> in X cross { 1 to 5 } without { <2,3> }
               with i > 5 and j < 2 do
                  sum <i,j,k> in X cross { 1 to 3 } cross Z do
                     p[i] * q[j] * w[j,k] >= if i == 2 then 17 else 53 end;
subto nonlin: 3 * x * y * z + 6 <= x^3 + z * y^2 + 3;
```

注意，在下列示例中，变量 i 和 j 是由 forall 语句确定的的，因此它们在每一轮 sum 表达式中的每次调用中都是固定不变的。

项式中的 *if* 语句

约束中的一部分项式通过放入 if-then-else-end 结构来根据条件构造。在这种特定情况下，else 部分是必需的，并且 then 与 else 的两个分支都必须在项式中包含某些变量项。

示例

```
subto c2: sum <i> in I :
     if (i mod 2 == 0) then 3 * x[i] else -2 * y[i] end <= 3;
```

约束中的 *if* 语句

可以通过 if 语句，在模型中设定具有两种不同形式的约束条件。在 if 的结果分支中也可以使用 forall 语句。其中，else 分支是可选的。

示例

```
subto c1: forall <i> in I do
  if (i mod 2 == 0) then  3 * x[i] >= 4
                    else -2 * y[i] <= 3 end;
```

使用 *and* 以合并约束

可以使用 and 将多个约束语句连接起来，从而将它们归为一组。这一组约束将始终被作为一个整体一同处理。该运算符在结合 forall 和 if 语句使用时尤其有用。

示例

```
subto c1: forall <i> in I:
  if (i > 3)
     then if (i == 4)
        then z[1] + z[2] == i
        else z[2] + z[3] == i
     end and
     z[3] + z[4] == i and
     z[4] + z[5] == i
  else
     if (i == 2)
        then z[1] + z[2] == i + 10
        else z[2] + z[3] == i + 10
     end and
     z[3] + z[4] == i + 10 and
     z[4] + z[5] == i + 10
  end;
```

约束属性

可以为约束指定特殊属性，以控制其在后续处理过程中的行为。

scale 在将约束写入文件之前，按 $1/\max|c|$ 的比例对其缩放，其中 c 表示该约束中所有系数的绝对值。

separate 不将该约束包含在初始线性规划中，而是在之后通过分离过程动态加入 [8]。该约束不需要用于可行性检查。当写入 LP 文件时，将被置于 `USER CUTS` 区域；在 SCIP 求解器中，该约束的 separate 属性将被设为 true。

checkonly 不将该约束包含在初始线性规划中，仅在之后用于可行性检查的分离过程中引入。当写入 LP 文件时，将被置于 `LAZY CUTS` 区域；在 SCIP 求解器中，该约束的 check 属性将被设为 true。

indicator 如果约束中包含 vif [9] (见下文示例)，那么该约束将被建模为一个指示约束 (indicator constraint)，而非采用 big-M 形式来表示。若使用了该属性，则无法再将模型导出为 MPS 格式文件，因为 MPS 格式不支持指示变量。

qubo 将该约束转换为一个二次型目标函数项。注意必要的二进制松弛变量会被自动创建。然而，到目前为止，整数变量不会被自动转换为二进制变量。

penaltyX 只能与 *qubo* 属性一起使用。将被转入目标函数的该约束乘上一个因子 $P = 10^X, X \in \{1, \ldots, 6\}$。

属性要写在每个约束末尾、分号之前，多个属性之间用逗号分隔。

示例

```
subto c1: 1000 * x + 0.3 * y <= 5, scale, checkonly;
subto c2: x + y + z == 7, separate;
```

[8] **译者注**：在一些规划建模问题当中，有一些约束是动态添置的。例如求解 TSP 时使用 DFJ 作为 ESCs 的情形。在这种情况下，需要使用 seperate 和 checkonly 属性标签。

[9] **译者注**：这类约束在数学规划中通常写作：if $y = 1$ then $a^T x \leqslant b$ else $y = 0$，一些求解器提供了原生的 vif 语法。

```
subto c3: vif x == 1 then y == 7 end, indicator;
subto c4: sum <i> in I : x[i] <= 15, qubo, penalty3;
```

特殊顺序集

在 ZIMPL 中可以为整数规划指定特殊顺序集[10] (SOS)。如果模型中包含任何 SOS ，则在生成 `lp` 或 `mps` 文件时，会同时输出一个后缀名为 `sos` 的文件。特殊顺序集的一般格式如下：

`sos` [名称]: [`type1`|`type2`] `priority` [表达式]: [线性表达式];

名称 可以是任意以字母开头的标识符，SOS 名称与约束名称共享命名空间。表达式 与目标函数中的表达式语法相同。`type1` 或者 `type2` 表示是类型一或是类型二的特殊顺序集。优先级 是可选参数，其含义与变量优先级设置相同。通过 `forall` 表述一次性生成多个特殊顺序集，参考如下示例。

示例

```
sos s1: type1: 100 * x[1] + 200 * x[2] + 400 * x[3];
sos s2: type2 priority 100 : sum <i> in I: a[i] * x[i];
sos s3: forall <i> in I with i > 2:
          type1: (100 + i) * x[i] + i * x[i-1];
```

扩展约束

ZIMPL 可以模仿条件约束的形式生成约束系。其一般的语法形式如下所示 (注意 `else` 的部分是可选的)：

`vif` 逻辑约束式 `then` 约束式 [`else` 约束式] `end`

其中 逻辑约束是由一个调用变量的线性表达式构成的。所有的这些变量都必须是含上下界的整型或者二元变量。在 逻辑表达式中，不得使用任何连续变量或者无限上下界的整型变量。所有的比较运算符 (<, ⩽, ==, !=, ⩾, >) 都是允许使用的。与此同时，多个语句也可以通过 `and` 、`or` 和 `xor` 关键字相连接，也可以通过 `not` 关键字进行取反。而在条件约束 (紧随在 `then` 或者 `else` 后的约束) 当中，可以包含具有上下界的连续型变量。需要注意，使用这一结构可能会生成若干个额外的约束和变量。

示例

```
var x[I] integer >= 0 <= 20;
subto c1: vif 3 * x[1] + x[2] != 7
   then sum <i> in I : y[i] <= 17
   else sum <k> in K : z[k] >= 5 end;
subto c2: vif x[1] == 1 and x[2] > 5 then x[3] == 7 end;
subto c3: forall <i> in I with i < max(I) :
   vif x[i] >= 2 then x[i + 1] <= 4 end;
```

扩展函数

可以在包含变量的项式上使用特殊的函数，这些函数将自动地被转换为一组不等式。这些函数的参数必须是仅由有上下界的整型变量或二元变量构成的线性项式。目前仅实现了计算项式 `t` 的绝对值的函

[10] **译者注**：另请参阅`https://en.wikipedia.org/wiki/Special_ordered_set`。

数 vabs(t)[11]，但其他函数 (如两项的最小值、最大值或一个项的符号函数) 可以类似的方式实现。同样，使用此构造将导致生成若干个额外的约束和变量。

示例

```
var x[I] integer >= -5 <= 5;
subto c1: vabs(sum <i> in I : x[i]) <= 15;
subto c2: vif vabs(x[1] + x[2]) > 2 then x[3] == 2 end;
```

[11] **译者注**：具体来说，该函数可以计算含变量的项式的绝对值。

6 建模示例

在本节中，我们讲通过若干示例，展示如何将一些众所周知的规划问题转述为 ZIMPL 的形式。

6.1 食谱问题

该问题源于 [Chv83, Chapter 1, page 3] 中的第一个示例。这是一个经典的食谱问题，有关其实际应用意义，可参见示例 [Dan90]。

给定一食物种类的集合 F 和一营养素的集合 N，有一张表 π_n 列举了每一种食物 f 中营养素 n 的含量。现给定参数 Π_n，定义为每种营养素的最低摄入量；Δ_f 表示每种食物可接受的最大份数。已知每种食物的价格 c_f，我们需要找到一个满足所有限制条件且总成本最低的膳食配择方案。为此，为每个 $f \in F$ 引入一个整型变量 x_f，用以表示提供食物 f 的份数。使用整型变量是因为只提供整份的食物，这也就是说，不能选择半份鸡蛋。则该问题可以被定义为：

$$\begin{aligned} \min \sum_{f \in F} c_f x_f & \qquad \text{subject to} \\ \sum_{f \in F} \pi_{fn} x_f \geqslant \Pi_n & \qquad \text{for all } n \in N \\ 0 \leqslant x_f \leqslant \Delta_f & \qquad \text{for all } f \in F \\ x_f \in \mathbb{N}_0 & \qquad \text{for all } f \in F \end{aligned}$$

将其转述为 ZIMPL 后如下所示：

```
set Food      := { "Oatmeal", "Chicken", "Eggs",
                   "Milk",    "Pie",     "Pork" };
set Nutrients := { "Energy",  "Protein", "Calcium" };
set Attr      := Nutrients + { "Servings", "Price" };

param needed[Nutrients] :=
   <"Energy"> 2000, <"Protein"> 55, <"Calcium"> 800;

param data[Food * Attr] :=
          |"Servings","Energy","Protein","Calcium","Price"|
|"Oatmeal"|         4 ,    110 ,       4 ,       2 ,     3 |
|"Chicken"|         3 ,    205 ,      32 ,      12 ,    24 |
|"Eggs"   |         2 ,    160 ,      13 ,      54 ,    13 |
|"Milk"   |         8 ,    160 ,       8 ,     284 ,     9 |
|"Pie"    |         2 ,    420 ,       4 ,      22 ,    20 |
|"Pork"   |         2 ,    260 ,      14 ,      80 ,    19 |;
#                       (kcal)      (g)      (mg) (cents)

var x[<f> in Food] integer >= 0 <= data[f, "Servings"];

minimize cost: sum <f> in Food : data[f, "Price"] * x[f];

subto need: forall <n> in Nutrients do
   sum <f> in Food : data[f, n] * x[f] >= needed[n];
```

满足所有要求的最便宜的一餐价格为 97 美分，包括四份燕麦片、五份牛奶和两份馅饼。

6.2 旅行商问题

本例展示如何生成对称旅行商问题 (TSP) 的指数级描述[12]，按 [Sch03, Section 58.5] 中的示例所述。

令 $G = (V, E)$ 为一完全连通图，其中 V 表示城市集合，E 表示城市间的连接边集合。为每条边 $(i,j) \in E$ 引入二元变量 x_{ij} 以指示边 (i,j) 是否被选择作为该环游的一部分，则 TSP 模型可表述为：

$$\begin{aligned}
\min \sum_{(i,j)\in E} d_{ij} x_{ij} \qquad & \text{subject to} \\
\sum_{(i,j)\in \delta_v} x_{ij} = 2 \qquad & \text{for all } v \in V \\
\sum_{(i,j)\in E(U)} x_{ij} \leqslant |U| - 1 \qquad & \text{for all } U \subseteq V, \emptyset \neq U \neq V \\
x_{ij} \in \{0,1\} \qquad & \text{for all } (i,j) \in E
\end{aligned}$$

相关数据从文件中读取，文件提供了城市的编号及其 x 和 y 坐标。假设城市间距离采用欧几里得距离计算。如下例所示：

```
# City       X    Y
Berlin     5251 1340
Frankfurt  5011  864
Leipzig    5133 1237
Heidelberg 4941  867
Karlsruhe  4901  840
Hamburg    5356  998
Bayreuth   4993 1159
Trier      4974  668
Hannover   5237  972
Stuttgart  4874  909
Passau     4856 1344
Augsburg   4833 1089
Koblenz    5033  759
Dortmund   5148  741
Bochum     5145  728
Duisburg   5142  679
Wuppertal  5124  715
Essen      5145  701
Jena       5093 1158
```

以下为 ZIMPL 中的模型表述。请注意，P[] 存储了城市集合的所有子集。因此，此方法最多仅能处理约 19 个城市规模的问题。有关如何求解更大规模实例的信息，请访问 CONCORDE 网站[13]。

```
set V             := { read "tsp.dat" as "<1s>" comment "#" };
set E             := { <i,j> in V * V with i < j };
set P[]           := powerset(V);
set K             := indexset(P);

param px[V]       := read "tsp.dat" as "<1s> 2n" comment "#";
param py[V]       := read "tsp.dat" as "<1s> 3n" comment "#";

defnumb dist(a,b) := sqrt((px[a]-px[b])^2 + (py[a]-py[b])^2);

var x[E] binary;

minimize cost: sum <i,j> in E : dist(i,j) * x[i, j];

subto two_connected: forall <v> in V do
```

[12]译者注：此处原文为“exponential description”，译者揣测其含义指是 TSP 的完整数学模型中的 DFJ ESCs 的数量随城市规模呈指数级增长。

[13]http://www.tsp.gatech.edu

```
   (sum <v,j> in E : x[v,j]) + (sum <i,v> in E : x[i,v]) == 2;

subto no_subtour:
   forall <k> in K with
      card(P[k]) > 2 and card(P[k]) < card(V) - 2 do
      sum <i,j> in E with <i> in P[k] and <j> in P[k] : x[i,j]
      <= card(P[k]) - 1;
```

输出的 LP 模型包含 171 个变量，239,925 个约束，且约束矩阵中具有 22,387,149 个非零元素，生成的 MPS 文件大小为 936 MB。CPLEX 在不到一分钟的时间内无需分支即求解至最优解[14]。

针对上述数据给出的一个最优化的环游方案是 Berlin, Hamburg, Hannover, Dortmund, Bochum, Wuppertal, Essen, Duisburg, Trier, Koblenz, Frankfurt, Heidelberg, Karlsruhe, Stuttgart, Augsburg, Passau, Bayreuth, Jena, Leipzig, Berlin。

6.3 含产能约束限制的设施选址问题

本节我们将给出*含产能约束限制的设施选址*问题的数学模型。该问题亦可视为一类具有装箱成本且箱型尺寸可变的*背包*问题，或视为考虑切割成本的*切割下料*问题。

给定一待建设的备选工厂集合 P 和一门店集合 S，各门店需求量为 δ_s 且必须被满足，需确定各工厂服务的门店的分配方案。建设工厂 p 的成本为 c_p，从工厂 p 运输货物至商店 s 的成本为 c_{ps}。每个工厂 p 仅有有限产能 κ_p。我们要求每个商店必须由唯一工厂服务，当然最终目标是寻求总成本最低的解决方案：

$$\begin{aligned}
\min \sum_{p\in P} c_p z_p + \sum_{p\in P, s\in S} c_{ps} x_{ps} \qquad & \text{subject to} \\
\sum_{p\in P} x_{ps} = 1 \qquad & \text{for all } s \in S && (2)\\
x_{ps} \leqslant z_p \qquad & \text{for all } s \in S, p \in P && (3)\\
\sum_{s\in S} \delta_s x_{ps} \leqslant \kappa_p \qquad & \text{for all } p \in P && (4)\\
x_{ps}, z_p \in \{0,1\} \qquad & \text{for all } p \in P, s \in S
\end{aligned}$$

我们设置二元变量 z_p，当且仅当工厂 p 被建设时，其取值为 1。此外，我们设置二元变量 x_{ps}，当且仅当工厂 p 为商店 s 提供服务时，其取值为 1。式 (2) 表示每个商店只能被分配给一个工厂[15]。不等式 (3) 确保为商店提供服务的工厂已被建设。不等式 (4) 保证为商店提供服务的工厂不会超出其产能限制。将此模型放进 ZIMPL 的代码里，即可得到下一页所示的例程。该程序所描述的问题实例的最优解是选择建设工厂 `A` 和 `C` 。商店 2、3 和 4 由工厂 `A` 服务，其余商店由工厂 `C` 服务。总成本为 1457。

[14]仅需 40 次单纯形迭代即可达到最优解。**译者注**：这个算例的求解似乎并不像本书中说的这么容易，笔者在自己的电脑 (ASUS Redolbook 14, AMD Ryzen™7 4700U, 16GB DDR4) 上运行 SCIP 求解该算例时遭遇了死机和系统崩溃的情况。特此警告！

[15]**译者注**：原文为“assigned to exactly one plant”，言下之意指，反过来，一个工厂可以服务多家门店。

```
set PLANTS := { "A", "B", "C", "D" };
set STORES := { 1 .. 9 };
set PS     := PLANTS * STORES;

# 各个工厂的建设费用和建成后各自的产能是多少？
param building[PLANTS]:= <"A"> 500, <"B"> 600, <"C"> 700, <"D"> 800;
param capacity[PLANTS]:= <"A">  40, <"B">  55, <"C">  73, <"D">  90;

# 各个门店的需求量
param demand  [STORES]:= <1> 10, <2> 14,
                         <3> 17, <4>  8,
                         <5>  9, <6> 12,
                         <7> 11, <8> 15,
                         <9> 16;

# 各个工厂到各门店的运输成本
param transport[PS] :=
      |  1,  2,  3,  4,  5,  6,  7,  8,  9 |
  |"A"| 55,  4, 17, 33, 47, 98, 19, 10,  6 |
  |"B"| 42, 12,  4, 23, 16, 78, 47,  9, 82 |
  |"C"| 17, 34, 65, 25,  7, 67, 45, 13, 54 |
  |"D"| 60,  8, 79, 24, 28, 19, 62, 18, 45 |;

var x[PS]     binary;  # 工厂p是否供应门店s？
var z[PLANTS] binary;  # 工厂p是否建成？

# 我们需要成本最低
minimize cost: sum <p> in PLANTS : building[p] * z[p]
             + sum <p,s> in PS : transport[p,s] * x[p,s];

# 每一家门店只能由一家工厂供应
subto assign:
  forall <s> in STORES do
    sum <p> in PLANTS : x[p,s] == 1;

# 服务工厂的门店必须先被建造出来
subto build:
   forall <p,s> in PS do
     x[p,s] <= z[p];

# 工厂的产能必须能够满足指派该厂供应的所有门店的总需求量
# The plant must be able to meet the demands from all stores
# that are assigned to it
subto limit:
   forall <p> in PLANTS do
     sum <s> in S : demand[s] * x[p,s] <= capacity[p];
```

6.4 The *n*-queens problem

The problem is to place n queens on a $n \times n$ chessboard so that no two queens are on the same row, column or diagonal. The n-queens problem is a classic combinatorial search problem often used to test the performance of algorithms that solve satisfiability problems. Note though, that there are algorithms available which need linear time in practice, like, for example, those of [SG91]. We will show four different models for the problem and compare their performance.

The integer model

The first formulation uses one general integer variable for each row of the board. Each variable can assume the value of a column, i. e., we have n variables with bounds $1 \ldots n$. Next, we use the `vabs` extended function to model an *all different* constraint on the variables (see constraint c1). This makes sure that no queen is located in the same column than any other queen. The second constraint (c2) is used to block all the diagonals of a queen by demanding that the absolute value of the row distance and the column distance of each pair of queens are different. We model $a \neq b$ by $\text{abs}(a - b) \geqslant 1$.

Note that this formulation only works if a queen can be placed in each row, i. e., if the size of the board is at least 4×4.

```
param queens := 8;

set C := { 1 .. queens };
set P := { <i,j> in C * C with i < j };

var x[C] integer >= 1 <= queens;

subto c1: forall <i,j> in P do vabs(x[i] - x[j]) >= 1;
subto c2: forall <i,j> in P do
             vabs(vabs(x[i] - x[j]) - abs(i - j)) >= 1;
```

The following table shows the performance of the model. Since the problem is modeled as a pure satisfiability problem, the solution time depends only on how long it takes to find a feasible solution.[16] The columns titled *Vars*, *Cons*, and *NZ* denote the number of variables, constraints and non-zero entries in the constraint matrix of the generated integer program. *Nodes* lists the number of branch-and-bound nodes evaluated by the solver, and *time* gives the solution time in CPU seconds.

Queens	Vars	Cons	NZ	Nodes	Time [s]
8	344	392	951	1,324	<1
12	804	924	2,243	122,394	120
16	1,456	1,680	4,079	>1 mill.	>1,700

As we can see, between 12 and 16 queens is the maximum instance size we can expect to solve with this model. Neither changing the CPLEX parameters to aggressive cut generation nor setting emphasis on integer feasibility improves the performance significantly.

[16] Which is, in fact, rather random.

The binary models

Another approach to model the problem is to have one binary variable for each square of the board. The variable is one if and only if a queen is in this square and we maximize the number of queens on the board.

For each square we compute in advance which other squares are blocked if a queen is placed on this particular square. Then the extended `vif` constraint is used to set the variables of the blocked squares to zero if a queen is placed.

```
param columns := 8;

set C   := { 1 .. columns };
set CxC := C * C;

set TABU[<i,j> in CxC] := { <m,n> in CxC with (m != i or n != j)
   and (m == i or n == j or abs(m - i) == abs(n - j)) };

var x[CxC] binary;

maximize queens: sum <i,j> in CxC : x[i,j];

subto c1: forall <i,j> in CxC do vif x[i,j] == 1 then
             sum <m,n> in TABU[i,j] : x[m,n] <= 0 end;
```

Using extended formulations can make the models more comprehensible. For example, replacing constraint c1 in line 13 with an equivalent one that does not use `vif` as shown below, leads to a formulation that is much harder to understand.

```
subto c2: forall <i,j> in CxC do
            card(TABU[i,j]) * x[i,j]
          + sum <m,n> in TABU[i,j] : x[m,n] <= card(TABU[i,j]);
```

After the application of the CPLEX presolve procedure both formulations result in identical integer programs. The performance of the model is shown in the following table. *S* indicates the CPLEX settings used: Either *(D)efault*, *(C)uts*[17], or *(F)easibility*[18]. *Root Node* indicates the objective function value of the LP relaxation of the root node.

Queens	S	Vars	Cons	NZ	Root Node	Nodes	Time [s]
8	D	384	448	2,352	13.4301	241	<1
	C				8.0000	0	<1
12	D	864	1,008	7,208	23.4463	20,911	4
	C				12.0000	0	<1
16	D	1,536	1,792	16,224	35.1807	281,030	1,662
	C				16.0000	54	8
24	C	3,456	4,032	51,856	24.0000	38	42
32	C	6,144	7,168	119,488	56.4756	>5,500	>2,000

[17] Cuts: `mip cuts all 2` and `mip strategy probing 3`.

[18] Feasibility: `mip cuts all -1` and `mip emph 1`

This approach solves instances with more than 24 queens. The use of aggressive cut generation improves the upper bound on the objective function significantly, though it can be observed that for values of n larger than 24 CPLEX is not able to deduce the trivial upper bound of n.[19] If we use the following formulation instead of constraint c2, this changes:

```
subto c3: forall <i,j> in CxC do
            forall <m,n> in TABU[i,j] do x[i,j] + x[m,n] <= 1;
```

As shown in the table below, the optimal upper bound on the objective function is always found in the root node. This leads to a similar situation as in the integer formulation, i. e., the solution time depends mainly on the time it needs to find the optimal solution. While reducing the number of branch-and-bound nodes evaluated, aggressive cut generation increases the total solution time.

With this approach instances, up to 96 queens can be solved. At this point, the integer program gets too large to be generated. Even though the CPLEX presolve routine is able to aggregate the constraints again, ZIMPL needs too much memory to generate the IP. The column labeled *Pres. NZ* lists the number of non-zero entries after the presolve procedure.

Queens	S	Vars	Cons	NZ	Pres. NZ	Root Node	Nodes	Time [s]
16	D	256	12,640	25,280	1,594	16.0	0	<1
32	D	1,024	105,152	210,304	6,060	32.0	58	5
64	D	4,096	857,472	1,714,944	23,970	64.0	110	60
64	C					64.0	30	89
96	D	9,216	2,912,320	5,824,640	53,829	96.0	70	193
96	C					96.0	30	410
96	F					96.0	69	66

Finally, we will try the following set packing formulation:

```
subto row: forall <i> in C do
   sum <i,j> in CxC : x[i,j] <= 1;

subto col: forall <j> in C do
   sum <i,j> in CxC : x[i,j] <= 1;

subto diag_row_do: forall <i> in C do
   sum <m,n> in CxC with m - i == n - 1: x[m,n] <= 1;

subto diag_row_up: forall <i> in C do
   sum <m,n> in CxC with m - i == 1 - n: x[m,n] <= 1;

subto diag_col_do: forall <j> in C do
   sum <m,n> in CxC with m - 1 == n - j: x[m,n] <= 1;

subto diag_col_up: forall <j> in C do
   sum <m,n> in CxC with card(C) - m == n - j: x[m,n] <= 1;
```

[19] For the 32 queens instance the optimal solution is found after 800 nodes, but the upper bound is still 56.1678.

Here again, the upper bound on the objective function is always optimal. The size of the generated IP is even smaller than that of the former model after presolve. The results for different instances size are shown in the following table:

Queens	S	Vars	Cons	NZ	Root Node	Nodes	Time [s]
64	D	4,096	384	16,512	64.0	0	<1
96	D	9,216	576	37,056	96.0	1680	331
96	C				96.0	1200	338
96	F				96.0	121	15
128	D	16,384	768	65,792	128.0	>7000	>3600
128	F				128.0	309	90

In case of the 128 queens instance with default settings, a solution with 127 queens is found after 90 branch-and-bound nodes, but CPLEX was not able to find the optimal solution within an hour. From the performance of the Feasible setting, it can be presumed that generating cuts is not beneficial for this model.

7 报错信息

以下是一份 ZIMPL 可能产生的无法理解的报错消息的 (希望是) 完整的列表：

101 Bad filename
通过-o 参数提供的文件名无效，可能是因为缺少文件名、目录名，或文件名以点开头。

102 File write error
写入输出文件时发生错误。错误描述将在下一行显示。有关错误含义，请参阅您的操作系统文档。

103 Output format not supported, using LP format
尝试选择了除 `lp` 、`mps` 、`hum` 、`rlp` 或 `pip` 之外的格式。

104 File open failed
尝试打开文件进行写入时发生错误。关于错误的描述将在下一行显示。有关错误信息的含义，请参阅您的操作系统文档。

105 Duplicate constraint name "xxx"
两个 `subto` 语句具有相同的名称。

106 Empty LHS, constraint trivially violated
约束的一侧为空，而另一侧不等于零。这种情况通常发生在对一个空的集合求和时。

107 Range must be $l \leqslant x \leqslant u$, or $u \geqslant x \geqslant l$
如果您指定范围，则必须在两侧使用相同的比较运算符。

108 Empty Term with nonempty LHS/RHS, constraint trivially violated
约束的中部为空，而约束的左侧或者右侧不等于 0。这种情况通常发生在对一个空的集合求和时。

109 LHS/RHS contradiction, constraint trivially violated
取值范围较低的一侧大于较高的一侧，例如 $15 \leqslant x \leqslant 2$。

110 Division by zero
你在尝试除以零。这不是一个好主意。

111 Modulo by zero
你在尝试让一个数字参与以 0 为模的运算，这将是不可行的。

112 Exponent value xxx is too big or not an integer
只允许对数值计算整数次幂，且幂指数不得超过二十亿。[20]

113 Factorial value xxx is too big or not an integer
你只能计算整数阶的阶乘。同时，计算一个超过二十亿的数的阶乘通常是一个坏主意。另请参阅报错 115。

114 Negative factorial value
你只能计算正数的阶乘。如果您需要为负数计算阶乘，请注意，对于所有偶数，结果将是正数，对于所有奇数，结果将是负数。

115 Timeout!
你在尝试计算一个大于 1000! 的数。另请参阅报错 112 的脚注。

116 Illegal value type in min: xxx only numbers are possible
你在尝试计算一些字符串的最小值。

[20]这种做法相当于是写了一个死循环 `for(;;)` ，或者更确切地来说就像是在递归 `void f(){f();}` 。

117 Illegal value type in max: xxx only numbers are possible
你在尝试计算一些字符串的最大值。

118 Comparison of different types
你在尝试比较不同类型的数据，例如比较数字和字符串。请注意，使用未定义的参数也可能导致此消息。

119 xxx of sets with different dimension
要对两个集合应用操作 xxx (取并集、差集、交集、对称差集)，它们必须具有相同维度的元组，这也就是说，每个元组必须包含相同个数的分量。

120 Minus of incompatible sets
要对两个集合应用操作 xxx (并集、差集、交集、对称差集) ，它们必须具有相同变量类型的元组，这也就是说，元组中的每个分量必须是相同的数据类型 (数字、字符串) 。

121 Negative exponent on variable
变量的指数为负数。不支持这种定义方式。

123 "from" value xxx is too big or not an integer
如果要生成集合，"from" 后的数字必须是一个绝对值小于二十亿的整数。

124 "upto" value xxx is too big or not an integer
如果要生成集合，"upto" 后的数字必须是一个绝对值小于二十亿的整数。

125 "step" value xxx is too big or not an integer
如果要生成集合，"step" 后的数字必须是一个绝对值小于二十亿的整数。

126 Zero "step" value in range
给定的 "step" (步长) 为零，因此永远无法达到 "upto" 的值。

127 Illegal value type in tuple: xxx only numbers are possible
`proj` 函数中调用的选择元组 (selection tuple) 只能包含数字。

128 Index value xxx in proj too big or not an integer
`proj` 函数的选择元组中的值不是整数或超过了二十亿。

129 Illegal index xxx, set has only dimension yyy
选择元组中的索引值大于被索引集合中元组的维度。

131 Illegal element xxx for symbol [21]
索引集初始化列表中使用的索引元组不是集合的索引集的成员。例如：`set A[{ 1 to 5 }] := <1> { 1 }, <6> { 2 };` 。

132 Values in parameter list missing, probably wrong read template
可能是 read 语句的模板形如`"<1n>"` ，只有一个元组，而不是 `"<1n> 2n"` 。

133 Unknown symbol xxx
使用了没有在作用域内定义的名称。

134 Illegal element xxx for symbol
初始化过程中给出的索引元组不是参数的索引集的成员。

135 Index set for parameter xxx is empty
尝试声明一个索引集为空的索引参数。很可能是索引集有一个 `with` 子句，它拒绝了所有元素。

[21] **译者注**：报错 130 在版本 2.02 中被移除，此处直接接继报错 131。

139 Lower bound for integral var xxx truncated to yyy (warning)
整型变量只能有整数边界，给定非整数边界则被调整为整数边界。

140 Upper bound for integral var xxx truncated to yyy (warning)
整型变量只能有整数边界，给定非整数边界则被调整为整数边界。

141 Infeasible due to conflicting bounds for var xxx
变量的上界小于下界。

142 Unknown index xxx for symbol yyy
符号的索引集不包含给定的索引元组。

143 Size for subsets xxx is too big or not an integer
给定的用于生成的子集的基数必须是一个小于二十亿的整数。

144 Tried to build subsets of empty set
用于生成子集的集合是空集。

145 Illegal size for subsets xxx, should be between 1 and yyy
用于生成的子集的基数必须在 1 和基集合的基数 (集合元素的个数) 之间。

146 Tried to build powerset of empty set
用于生成幂集的集合是空集。

147 use value xxx is too big or not an integer
use 参数的值必须是一个小于二十亿的整数。

148 use value xxx is not positive
use 参数不能为负数或 0。

149 skip value xxx is too big or not an integer
skip 参数的值必须是一个小于二十亿的整数。

150 skip value xxx is not positive
skip 参数不能为负数或 0。

151 Not a valid read template
read 模板必须是类似"<1n,2n>" 的格式，按顺序必须有 < 和 > 。

152 Invalid read template syntax
除了任何分隔符 (如< 、> 和逗号) 之外，模板必须由数字字符对组成，如 1n, 3s 。

153 Invalid field number xxx
模板中的字段编号必须在 1 到 255 之间。

154 Invalid field type xxx
字段类型只能是 n 和 s 。

155 Invalid read template, not enough fields
分隔符之间至少要有一个字段。

156 Not enough fields in data
模板指定的字段编号高于数据中实际找到的字段数。

157 Not enough fields in data (value)
模板指定的字段编号高于数据中实际找到的字段数。错误发生在值字段中的索引元组之后。

159 Type error, expected xxx got yyy
找到的数据类型不是预期的类型，例如从数字中减去字符串将导致此消息。

160 Comparison of elements with different types xxx / yyy
比较两个不同元组中的元素时，发现它们的变量类型不同。

161 Line xxx: Unterminated string
该行具有奇数个" 字符。某一字符串已开始，但未封闭。

162 Line xxx: Trailing "yyy" ignored (warning)
文件中的最后一个分号后发现有内容。

163 Line xxx: Syntax Error
某一语句未以如下关键字开头：set, param, var, minimize, maximize, subto, or do

164 Duplicate element xxx for set rejected (warning)
尝试向集合中添加一个已存在的元素。

165 Comparison of different dimension sets (warning)
比较两个集合，但它们的维度元组不同。(这意味着除了空集之外，它们永远也不可能相等。)

166 Duplicate element xxx for symbol yyy rejected (warning)
尝试向 yyy 中添加一个已存在的元素。

167 Comparison of different dimension tuples (warning)
尝试比较两个维度不同的元组。

168 No program statements to execute
在加载的文件中没有找到 ZIMPL 语句。

169 Execute must return void element
这不应该发生。如果您遇到此错误，请将.zpl 文件发送至mailto:koch@zib.de。

170 Uninitialized local parameter xxx in call of define yyy
调用定义 (define) 时，其中一个参数是一个 "名称" (变量的名称)，但没有为其定义值。

171 Wrong number of arguments (xxx instead of yyy) for call of define zzz
调用定义 (define) 时，参数数量与定义的不同。

172 Wrong number of entries (xxx) in table line, expected yyy entries
在参数的初始化表中，每一行输入的条目 (entries) 的个数必须与表的索引行 (第一行) 相同。

173 Illegal type in element xxx for symbol
一个参数表只能有一个数据类型，要么是数字，要么是字符串。在初始化过程中同时指定了两种类型。

174 Numeric field xxx read as "yyy". This is not a number
试图读取一个在模板中指定为 'n' 的字段，但读取的内容不是有效的数字。

175 Illegal syntax for command line define "xxx" – ignored (warning)
使用命令行选项 -D 指定的参数必须匹配 name=value 的格式。其中 name 必须是一个合法的标识符，这也就是说，它必须以字母开头，并且只能由字母、数字和下划线组成。

176 Empty LHS, in Boolean constraint (warning)
(逻辑约束的) 左半边为空，这也就是说，包含变量的项式为空。

177 Boolean constraint not all integer
在逻辑约束中，不允许出现连续型 (实型) 变量。

178 Conditional always true or false due to bounds (warning)
由于变量的上下界，逻辑约束的全部或部分始终为真或者假。

179 Conditional only possible on bounded constraints
在某个逻辑约束中至少存在一个变量没有有限边界。

180 Conditional constraint always true due to bounds (warning)
条件约束的结果部分恒为真。这是由于包含了变量的上下界。

181 Empty LHS, not allowed in conditional constraint
条件约束的结果部分不能为空。

182 Empty LHS, in variable vabs
`vabs` 函数的参数当中没有变量。要么所有的值都会被归零，要么只需使用 `abs` 即可。

183 vabs term not all integer
`vabs` 函数的声明中包含了非整型变量。由于数值的原因 (numerical reason)，不允许将连续型变量作为参数传递给 `vabs` 。

184 vabs term not bounded
`vabs` 函数内的项式中至少包含一个无界变量。The term inside a `vabs` has at least one unbounded variable.

185 Term in Boolean constraint not bounded
`vif` 内的项式中至少包含一个无界变量。

186 Minimizing over empty set – zero assumed (warning)
取极小值时的索引表达式为空，因为该表达式的值为零。[22]

187 Maximizing over empty set – zero assumed (warning)
取极大值时的索引表达式为空，因为该表达式的值为零。

188 Index tuple has wrong dimension
索引元组中的元素数量与被索引集合的元组索引的维度不一致。导致此错误的原因，既可能是因为在参数初始化列表中，条目的索引元组的维度与该参数的索引集合维度不同；也可能是因为在集合初始化列表中，条目的索引元组维度与该集合的索引集的维度不同。如果您使用 `powerset` 或 `subset` 指令，则索引集必须是一维的。

189 Tuple number xxx is too big or not an integer
给定的元组的编号必须是小于二十亿的整数。

190 Component number xxx is too big or not an integer
给定的分量的编号必须是小于二十亿的整数。

191 Tuple number xxx is not a valid value between 1..yyy
元组的编号必须在 1 和集合的基数之间。

192 Component number xxx is not a valid value between 1..yyy
分量的编号必须在 1 和集合的维数之间。

[22] **译者注**：我确实不知道这句话是什么意思，所以此处是照着字面翻译的。

193 Different dimension tuples in set initialization
同一列表中的元组的维度不同。

195 Genuine empty set as index set (warning)
某一索引集索引的被索引集合恒为空集。

197 Empty index set for set
某一集合的索引集为空集。

198 Incompatible index tuple
给定的索引元组具有固定的分量，该分量的数据类型与集合中的元组的同一分量的数据类型不一致。

199 Constants are not allowed in SOS declarations
在声明 SOS 时，权重只能与变量一同使用。单独使用的权重没有意义。

200 Weights are not unique for SOS xxx (warning)
特殊有序集中分配给变量的所有权重都必须是唯一的。

201 Invalid read template, only one field allowed
读取一个单一的参数值时，read 模板必须只包含一个单一的字段规则。

202 Indexing over empty set (warning)
索引集是空的。

203 Indexing tuple is fixed (warning)
某个索引表达式产生的索引集是一个固定的集合。其结果是只有这一个 (对应的) 元素会被搜索到。

204 Random function parameter minimum= xxx >= maximum= yyy
`random` 函数的第二个参数必须严格大于第一个参数。

205 xxx excess entries for symbol yyy ignored (warning)
读取符号 `yyy` 中的数据时，文件中存在的条目数比符号的索引数多 `xxx` 个。多余的条目会被忽略。

206 argmin/argmax over empty set (warning)
`argmin` 或 `argmax` 的索引表达式为空。返回的结果为空集。

207 "size" value xxx is too big or not an integer
`argmin` 和 `argmax` 函数的 "size" 参数必须是一个绝对值小于二十亿的整数。

208 "size" value xxx not >= 1
`argmin` 和 `argmax` 函数的 "size" 参数最小为 1，因为它表示结果集合的最大基数。

209 MIN of set with more than one dimension
表达式 `min(A)` 中集合 A 的元素只能由包含数字的一维元组组成。

210 MAX of set with more than one dimension
表达式 `max(A)` 中集合 A 的元素只能由包含数字的一维元组组成。

211 MIN of set containing non number elements
表达式 `min(A)` 中集合 A 的元素只能由包含数字的一维元组组成。

212 MAX of set containing non number elements
表达式 `max(A)` 中集合 A 的元素只能由包含数字的一维元组组成。

213 More than 65535 input fields in line xxx of yyy (warning)
文件 yyy 中的第 xxx 行的输入字段超过 65535 个。在数据中插入一些新行！[23]

214 Wrong type of set elements – wrong read template?
很可能您在模板中尝试使用 "n+" 而不是 "<n+>" 从流 (stream) 中读取集合。

215 Startvals violate constraint, ... (warning)
如果给定的 startvals 相加，它们就会违背约束。报错信息中给出了 LHS 和 RHS 的相关错误详情。

216 Redefinition of parameter xxx ignored
同名的参数被声明了两次。典型的用法是在 ZIMPL 文件里面声明参数的默认值，再通过命令行定义的方式覆盖它们。

217 begin value xxx in substr too big or not an integer
substr 函数的 begin 参数必须是一个绝对值小于二十亿的整数。

218 length value xxx in substr too big or not an integer
substr 函数的 length 参数必须是一个绝对值小于二十亿的整数。

219 length value xxx in substr is negative
substr 函数的 length 参数必须大于或等于 0。

220 Illegal size for subsets xxx, should be between yyy and zzz
要生成子集的基数必须在指定的下界和基集合的基数之间。

222 Term inside a then or else constraint not linear [24]
then 或 else 中的项式不是线性的形式。The term inside a then or else is not linear.

223 Objective function xxx overwrites existing one (warning)
遇到了另一个目标函数形式的声明，覆盖了前一个。

301 variable priority has to be integral (warning)
如果给定了变量的分支优先级，则这些优先级必须是整数。

302 SOS priority has to be integral (warning)
如果给定了 SOS 的优先级，则这些优先级必须是整数。

401 Slack too large (xxx) for QUBO conversion 该约束所需的松弛变量的上界超过了 2^{30}。

403 Non linear term can't be converted to QUBO
只有线性项式可以自动转化为 QUBO。

404 Non linear expressions can't be converted to QUBO
只有不含指示约束的线性表达式可以被自动转化为 QUBO。

600 File format can only handle linear and quadratic constraints (warning)
所选的文件格式只能处理线性和二次项约束。更高阶的约束将被忽略。

601 File format can only handle binary variables (warning)]
所选择的文件格式只能处理二元变量。其他数据类型的变量将被忽略。

602 QUBO file format can only handle linear and quadratic term (warning)
根据定义，QUBO 的文件格式只能处理线性和二次的项式。无法写入文件。

[23]**译者注**：意指换行。

[24]**译者注**：文档源码中的报错 221 被原作者注释掉了。

700 log(): OS specific domain or range error message
调用 log 函数时参数为零或者负数，或者由于参数太小，无法表示为 double 数据类型。

701 sqrt(): OS specific domain error message
调用 log 函数时参数为负数。

702 ln(): OS specific domain or range error message
调用 log 函数时参数为零或者负数，或者由于参数太小，无法表示为 double 数据类型。

800 parse error: expecting xxx (or yyy)
解析错误。找到的内容与预期不符，您输入的语句无效。

801 Parser failed
解析例程失败。这种情况不应该发生。如果您遭遇了此错误，请将此 .zpl 文件发送电子邮件到 mailto:koch@zib.de。

802 Regular expression error
read 语句的 match 参数中给出的一个正则表达式无效。有关错误详情，请参阅错误信息。

803 String too long xxx > yyy
程序遭遇了一个大于 1 GB 的字符串。

900 Check failed!
某个 check 指令的评估结果不为“真”。

参考文献

[Chv83] Vašek Chvátal. *Linear Programming*. H.W. Freeman and Company, New York, 1983.

[Dan90] Georg B. Dantzig. The diet problem. *Interfaces*, 20:43–47, 1990.

[FGK03] R. Fourier, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. Brooks/Cole—Thomson Learning, 2nd edition, 2003.

[GNU03] GNU multiple precision arithmetic library (GMP), version 4.1.2., 2003. Code and documentation available at http://gmplib.org.

[IBM97] IBM optimization library guide and reference, 1997.

[ILO02] ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. *ILOG CPLEX 8.0 Reference Manual*, 2002. Information available at http://www.ilog.com/products/cplex.

[Kal04a] Josef Kallrath. Mathematical optimization and the role of modeling languages. In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages 3–24. Kluwer, 2004.

[Kal04b] Josef Kallrath, editor. *Modeling Languages in Mathematical Optimization*. Kluwer, 2004.

[Koc04] Thorsten Koch. *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin, 2004.

[Sch03] Alexander Schrijver. *Combinatorial Optimization*. Springer, 2003.

[Sch04] Hermann Schichl. Models and the history of modeling. In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages 25–36. Kluwer, 2004.

[SG91] Rok Sosič and Jun Gu. 3,000,000 million queens in less than a minute. *SIGART Bulletin*, 2(2):22–24, 1991.

[Spi04] Kurt Spielberg. The optimization systems MPSX and OSL. In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, pages 267–278. Kluwer, 2004.

[vH99] Pascal van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, Massachusetts, 1999.

[XPR99] *XPRESS-MP Release 11 Reference Manual*. Dash Associates, 1999. Information available at `http://www.dashoptimization.com`.