.

AURORA DEL CAMP – SPECIFICATION

JULIAN PFEIFLE

This document outlines some of the issues that arise in the implementation of Gilad's proposal [1].

Given the availability of powerful free and open source solvers for integer programs such as CBC [2], it seems natural to pursue an integer programming formulation. Of course, free solvers are not as good as the best commercial ones, but the most recent benchmarks [3] indicate that CBC is reasonably competitive; more precisely, it's the most competitive among all solvers that have an open source license (in the case of CBC, the "Eclipse Public License") that permits Gilad to use it commercially without paying any license fees.

1. Problem formulation

We start by translating Gilad's document into the language of integer programs.

1.1. **Variables.** We work with a set C of crops, the set $W = \{0, \dots, 51\}$ of weeks in the year, and a set A of "unit lots", i.e., indivisible units of farmland dedicated to a single crop or task. The set A has a distinguished member $a_0 \in A$ that stands for off-field work. In this fictitious area of lot, any number of tasks may be performed simultaneously, while only one thing at a time may be done in all lots $A \setminus a_0$.

We will have several different families of variables, one family for each specific task. These are further subdivided into families that only affect the field, and are thus common to all crops, and those that are specific to each crop.

Tasks common to all crops: ti for tilling, rv for rotovating, gm for green manure planting, ft for fertilizing, bb for bed building, si for setting up irrigation, sr for setting rows, we for weeding.

Tasks specific to a crop: by for buying seeds, ss for soaking seeds, cs for cutting or separating cloned seeds, gc for false germination and cleaning, pl for planting[1], fu for fumigating, th for thinning, tr for trimming, co for covering, ha for harvesting.

Each of the "common" families contains variables indexed by the week $w \in W$ of the year, and the piece of land $a \in A$. For instance, the tilling variables are $ti = \{ti_{w,a} : w \in W, a \in A\}$. The "crop-specific" families, on the other hand, contain more variables, because they also depend on the type of crop $c \in C$. For example, the set of seed-soaking variables is $ss = \{ss_{c,w,a} : c \in C, w \in W, a \in A\}$. Each individual variable in each of these families can take on the value 0 or 1, depending on whether or not the task at hand is undertaken for crop c in the unit lot a during week w.

With an estimated 40 types of crops and 40 unit lots, we get an upper bound of

$$8 \times 40 \times 52 \text{ (common)} \;+\; 10 \times 40 \times 40 \times 52 \text{ (specific)} \;=\; 848\,640 \text{ variables.}$$

One the one hand, this is well within the reach of commercial solvers such as Gurobi (we'll have to see how cbc performs, though); on the other, there will actually be substantially less variables than this because not all crops need all variables. For example, a crop that is bought as a seedling from a nursery does not need variables from the families by, ss, cs, gc.

Date: Version of November 17, 2011.

[1] We consider transplanting and planting to be the same process.

1.2. **Constraints.** To formulate our constraints, we need some additional data on our crops:

Plantation interval pi_c: How many weeks must pass, at least, from one planting of crop c to the next

Yield $y_{c,i}$: The amount of fruit that crop c yields i weeks after planting, for $0 \leq i \leq pi_c$.

Now we can formulate the constraints:

(1) *In each week, there is at most one crop planted at each unit lot:*

$$\sum_{c \in C} pl_{c,w,a} \ \leq \ 1 \qquad \text{for all } w \in W \text{ and } a \in A$$

(2) *Respect plantation intervals:*

$$\sum_{i=w}^{w+pi_c} pl_{c,i,a} \ \leq \ 1 \qquad \text{for all } w \in W, c \in C, a \in A, \tag{1.1}$$

because during any interval of pi_c consecutive weeks, there may occur at most one planting. Here and throughout, index additions such as $i + pi_c$ are understood to be taken modulo 52.

(3) *Prescribe the yield:* If we want the total yield of crop c in a given week $w \in W$ to be $Y_{c,w}$, we must impose

$$\sum_{a \in A} \sum_{i=0}^{pi_c} y_{c,i} \, pl_{c,w-i,a} \ = \ Y_{c,w},$$

because a crop planted i weeks before week w has a yield of $y_{c,i}$ in week w. This works because by the foregoing constraint (1.1), at most one of the variables $\{pl_{c,w-i,a} : 0 \leq i \leq pi_c\}$ can have the value 1, while the rest must be 0.

(4) *Precedence constraints:* Here we must deal with the fact that we model our year to be cyclic, i.e., the same 52 weeks repeat again and again, as witnessed by our index addition modulo 52 in condition (1.1). Nevertheless, some tasks must be completed before others can start, which implies a linear and not circular ordering of time. We model this by interpreting precedence constraints to be valid only within a half-year horizon, so that *"x must happen before y"* is interpreted as meaning *"x may not happen until half a year after y"*.

For example, the constraint that "tilling" must happen before "weeding" is modeled as

$$ti_{w+i,c} \ \leq \ we_{w,c} \qquad \text{for all } w \in W, c \in C, \text{ and } i \in \{0, \dots, 26\}.$$

this is still incorrect To check this, note that the inequality is not satisfied, for example, if $ti_{4,c} = 1$ and $we_{0,c} = 1$ (because these variables say we till in week 4, after having weeded in week 0), but is satisfied if $ti_{0,c} = 1$ and $we_{4,c} = 1$.

1.3. **Objective function.** Each crop $c \in C$ has a yield of $y_{c,w}$, depending on the week $w \in W$ it is planted. The objective function we want to maximize is thus

$$f \ = \sum_{c \in C, y \in Y} y_{c,w} \sum_{a \in A} x_{c,w,a}$$

2. Server-side technology

Gilad's intention is to make the program available on a server. That's fine, except that we need to be able to install c++ and cbc on such a server.

REFERENCES

[1] G. BUZI, *Aurora Del Camp Crop Planner — a small farm plans big*, November 2011.

[2] *Cbc (coin-or branch and cut), an open-source mixed integer programming solver written in C++*. `https://projects.coin-or.org/Cbc`.

[3] H. D. MITTELMANN, *Performance of optimization software — an update*. `http://plato.asu.edu/talks/mittelmann_bench.pdf`, November 2011.