Aplikace matematiky

Karel Prokop; Štefan Chochol

Algorithms. 48. TRANSPORT. Systems of material flow

Aplikace matematiky, Vol. 28 (1983), No. 2, 156–160

Persistent URL: http://dml.cz/dmlcz/104019

48. TRANSPORT

SYSTEMS OF MATERIAL FLOW

Dr. KAREL PROKOP, CSc., Dr. ŠTEFAN CHOCHOL

Výzkumný ústav zemědělské techniky, 163 07 Praha 6 - Řepy

The system considered has steady elements, *points*, and moving elements, *machines*. There are two sorts of points and machines in the system. A *source* is a point where (continuous) material is produced and a *sink* is a point where material is absorbed. A *vehicle* is a machine used for transporting material from point to point and a *loader* is a machine used for loading or unloading vehicles with material at a given point. At every point there is a queue *vehicles* and a queue *loaders*. The machines *enter* the queues according to regimes *seizel* and *seizev*, respectively. The points are connected by *branches* which represent ways; every branch leading from a certain point belongs to its set *contacts* and has two attributes: its *endpoint* and a *distance* between this endpoint and the point from which the branch leads.

Every vehicle moves along branches and stops at points. It is released from a point if the condition *ready* is satisfied which means that the vehicle is either *full* or *empty* according to whether the point is a source or a sink. If a vehicle stops at a point it is gradually served by some loaders, the service being loading in case the point is a source or unloading if the point is a sink. The vehicle must *wait* in the queue *vehicles* if all loaders are busy. Every vehicle has its *total* of material, its *capacity* (maximum possible total), its *route* through the system and its procedure *move* which determines travelling along various branches. The route contains *stops* representing points to be met. The latest stop of a vehicle is referenced by its attribute *pointer*. Vehicles can be e.g. trucks, ships, trains etc.

A loader is either a simple loader (*siloader*) having some *rate* of transport (e.g. moving chain transporting material from or to a vehicle) or a loader with regeneration (*regloader*) which has a reservoir of certain *volume* that must be either refilled or reemptied, respectively, during some time called *regeneration*. As an example of a regloader we can give a combine harvester or a sowing machine. There is an interaction (lasting *duration* time units) between a loader and a vehicle during which the former will *change* the latter by adding a *portion* of material to the vehicle's total. Regloader's duration and portion are equal to its *loadduration* and $\pm$ *volume*,

respectively. When there are no vehicles at disposal, the loaders *wait* in the queue *loaders*.

The machines waiting in queues *record* the total *waiting* time; the vehicles must distinguish between the waiting time for loading and that for unloading, the latter being called *unldwaiting*. The distinguishing is performed at points. Every machine has its *place* referring to the point where this machine is actually placed. If one forms a system then every machine X must be placed at some point P by a statement *P. seize* (X); the activity of the system is started by the activation of all loaders at some source P by a statement *P. start*;

```
SIMULATION class TRANSPORT;

begin
class point; virtual: procedure seizel, seizev, start, record;
                      boolean procedure ready;
begin ref (head) loaders, vehicles, contacts;
      procedure seize (X); ref (machine) X;
        begin X. place :- this point; X. enter (this point) end;
loaders :- new head; vehicles :- new head; contacts :- new head
end ** of point **;
point class source;
begin procedure start;
   L: inspect loaders. first when loader do
      begin activate this loader; go to L end;
      procedure record (V); ref (vehicle) V; V. record;
      boolean procedure ready; ready := current qua vehicle. full;
end ** of source **;
point class sink;
begin boolean procedure ready; ready := current qua vehicle. empty;
      procedure record (V); ref (vehicle) V;
      inspect V do unldwaiting := unldwaiting + time - waitstart;
end ** of sink **;
process class machine; virtual: procedure enter;
begin real waiting, waitstart; ref (point) place;
      procedure record; waiting := waiting + time - waitstart;
      procedure wait;
      begin waitstart := time; enter (place); passivate end;
end ** machine **;
link class branch (endpoint, distance);
        ref (point) endpoint; real distance;
link class stop (object); ref (point) object;
machine class loader;
```

```
        virtual: real procedure portion, duration, regeneration;
begin procedure enter (P); ref (point) P; P. seizel (this loader);
      procedure change (V); ref (vehicle) V;
                inspect V do total := total + portion (V);
out; inner; while true do
begin hold (regeneration);
inspect place. vehicles.first when vehicle do
begin out; place. record (this vehicle); hold (duration (this vehicle));
  change (this vehicle); activate this vehicle
end otherwise wait end ** of loader **;
loader class regloader (volume, loadduration);
real volume, loadduration;
begin real procedure duration(V); ref (vehicle) V;
duration := loadduration;
end ** of regloader **;
regloader class rgldso;
begin real procedure portion(V); ref (vehicle)V;
                      portion := volume,
      if place in sink then ERROR;
end ** of rgldso **;
regloader class rgldsi;
begin real procedure portion(V); ref (vehicle)V;
                      portion := -volume;
      if place in source then ERROR;
end ** of rgldsi **;
loader class siloader (rate); real rate;
begin real procedure regeneration; regeneration := 0;
      real procedure duration(V); ref (vehicle)V;
          duration := rate × abs (portion(V));
      rate := 1/rate
end ** of siloader **;
siloader class sildso;
begin real procedure portion(V); ref (vehicle)V;
                      inspect V do portion := capacity - total;
      if place in sink then ERROR;
end ** sildso **;
siloader class sildsi;
begin real procedure portion(V); ref (vehicle)V; portion := -V. total;
      if place in source then ERROR;
end ** of sildsi **;
machine class vehicle (capacity); real capacity;
      virtual: boolean procedure empty, full; procedure move;
```

```
begin real unldwaiting, total; ref (head) route; ref (stop) pointer;
      procedure enter(P); ref (point)P; P. seizev (this vehicle);
      pointer :- route. first; if pointer = = none then ERROR;
      while true do
      begin while ¬ place. ready do
        inspect place. loaders. first when loader do
        begin out; record; hold (duration (this vehicle));
              change (this vehicle); activate this loader
        end otherwise wait;
begin ref (point) p1; ref (stop) pt; pt :- pointer. suc;
      if pt = = none then pt :- route. first;
      p1 :- pt. object; move (p1); place :- p1; pointer :- pt
end end end ** of vehicle **; end ** of TRANSPORT **;
```

Let us introduce an example which is often met in agriculture [1]: the system has one *pair* of points, the first one being a source *psource*, where all R loaders are regloaders, and the second one is a sink *psink*, where all S loaders are siloaders. There are V vehicles moving through the system. The regeneration time of loaders and the traveling time of vehicles are random numbers normally distributed, other values being constant; all parameters are read from the computer input. The regime of every queue is FIFO except that of the vehicles at the *psource* where the ordering is according to the vehicle's total. A vehicle is *full* if its total differs from its capacity by less than the maximum volume of the regloaders. This quantity is called *safetyvol*. A vehicle is *empty* when its total is less than *minvol*, the minimum volume of regloaders. Let us mention that the procedure *move* need not use contacts which can be let empty.

```
TRANSPORT class TR(R, S, V, U, U1, U2, simperiod);
              integer R, S, V, U, U1, U2; real simperiod;
begin integer I, J; real safetyvol, minvol; ref (head) pair;
      ref (point) psink, psource; ref(vh)Y;
rgldso class rgld (p1, p2); real p1, p2;
begin real procedure regeneration;
regeneration := normal (p1, p2, U); end;
vehicle class vh (lp1, lp2, gp1, gp2); real lp1, lp2, gp1, gp2;
begin procedure move(P); ref (point)P;
      hold (if place in source then normal (lp1, lp2, U1)
                               else normal (gp1, gp2, U2));
      boolean procedure full; full := total > capacity - safetyvol;
      boolean procedure empty; empty := total < minvol;
end ** of vh **;
source class solp;
begin procedure seizel(L); ref (loader)L; L. into (loaders);
```

```
        procedure seizev(V); ref (vehicle)V;
          begin ref (vehicle)X, XX; V. into (vehicles); X :- XX :- V;
                for X :- X. pred while X =/= none do
                if X. total < V. total then XX :- X else go to exit;
                exit : if XX =/= V then V. precede (XX)
          end;
end ** of solp **;
sink class silp;
begin procedure seizel(L); ref (loader)L; L. into (loaders);
      procedure seizev(V); ref (vehicle)V; V. into (vehicles);
end ** of silp **;
pair :- new head; psource :- new solp; new stop (psource). into (pair);
psink :- new silp; new stop (psink). into (pair); minvol := 1000000.
for I := 1 step 1 until S do
psink. seize (new sildsi (inreal));
for I := 1 step 1 until V do
inspect new vh (inreal, inreal, inreal, inreal, inreal) do
        begin psource. seize (this vh); route :- pair end;
for I := 1 step 1 until R do
inspect new rgld (inreal, inreal, inreal, inreal,) do
begin psource. seize (this rgld);
      if volume > safetyvol then safetyvol := volume;
      if volume < minvol then minvol := volume
end;
for Y :- psource. vehicles. first, Y. suc while Y=/=none do
if capacity < safetyvol then ERROR;
psource. start; hold (simperiod)
end *** of TR ***;
```

The classes presented were tested at IBM 370/138, ICL 4−72 and CDC 3300 computers and used for optimization of the plant production in the Czechoslovak agricultural farms and cooperatives. Simulation of one shift of grain harvest with two combine harvesters, two trucks and one belt transporter took approximately 2 seconds of CPU-time.

Acknowledgement. The authors thank Dr. Evžen Kindler, Dr. Marek Malík and Ing. Antonín Mojka for their effective help and stimuli during the development of the presented classes.

References

[1] *Š. Chochol, K. Prokop, M. Malík, E. Kindler:* Užití jazyka Simula 67 při řešení některých problémů mechanizované zemědělské výroby (Use of Language SIMULA 67 for solution of some problems of mechanized agricultural production — in Czech). In: Simulace systémů '78, Dům techniky ČSVTS, Ostrava 1978, pp. 212—217.