HIGH MULTIPLICITY SCHEDULING PROBLEMS

Yves CRAMA

HEC Management School

University of Liège

Francqui Lecture, KUL, April 2010

Outline

- What is a high-multiplicity scheduling problem?
- Complexity analysis of HMSP
- Flowshops with flexible operations
- Just-In-Time sequencing
- High-multiplicity traveling salesman problem

What is a HMSP? (1)

Usual input of a (one machine) scheduling problem is:

- Number of jobs $1,2,\ldots,n$
- For each job j, a list of attributes like
  - Processing time p_j
  - Release date r_j
  - Due date d_j
  - etc.

What is a HMSP? (2)

=> input size:

$$O(n\ L),$$

where L is the encoding size of the attributes.

What is a HMSP? (3)

In certain applications, jobs are distributed in a small number of classes and all jobs in a same class are identical.

=> Input :

- number of classes s
- number of jobs n_i in class i ($i = 1,2,\ldots,s$)
- attributes of a representative job in class i

E.g., for $s = 1$: identical jobs

Example: low-multiplicity

s = 250 jobs:

p_i	d_i
3	24
1	6
4	15
2	12
6	9
3	17
5	11
4	23 ...

Example: high-multiplicity

s = 4 types of jobs:

p_i	d_i	n_i
3	24	50
1	6	100
4	15	75
2	12	75

What is a HMSP? (4)

=> input size:

$$O(s \log n + s L)$$

instead of

$$O(n L)$$

(where L is the encoding size of the attributes).

This is much more compact if $s << n$ or if s is constant.

What is a HMSP? (5)

In particular,

• a problem which is polynomially solvable with low-multiplicity input can be solved in pseudo-polynomial time, but not necessarily in polynomial time, with HM input;

• not even easy to prove that a HMSP is in *NP* (because a natural certificate is a schedule, which is exponentially long in the input size).

Example : Cyclic manufacturing

- s types of products have to be produced in large numbers (say, infinitely many units)
- production ratios are fixed: e.g.

  $(r_1, r_2, \ldots, r_s) = (1/2, 1/4, 1/8, 1/8)$
- a Minimal Part Set (MPS) is a minimal batch of products which satisfies the target ratios and which can therefore be cyclically produced; e.g., MPS = (4, 2, 1, 1).

Example : Cyclic manufacturing

- In order to describe an instance, it is sufficient to give the MPS $(n_1, n_2, \ldots, n_s)$ and the characteristics of each part type i.

Example: multiprocessor scheduling

- m parallel machines
- available makespan: B
- s job classes
- n_i jobs in class i ($i = 1,2,\ldots,s$)
- processing time p_i in class i

Is there a feasible schedule ?

Example: multiprocessor scheduling

Case $s = 2$:

- m parallel machines
- available makespan: B
- n_1 jobs of length p_1, n_2 jobs of length p_2

(6 numbers !)

McCormick, Smallwood, Spieksma (2001) give a polynomial algorithm for this case.

Case $s = 3$ is open (progress by Agnetis et al.)

Early work (1)

- Rothkopf, *Operations Research* (1966)
- Psaraftis, *Operations Research* (1980)
- Cosmadakis and Papadimitriou, *SIAM J. Computing* (1984)
- Hochbaum and Shamir, *Discrete Applied Math*. (1990), *Operations Research* (1991)
- Shallcross *OR Letters* (1992)

Early work (2)

Hochbaum and Shamir coined the term "high multiplicity problems".

They observed explicitly that, since the input size is

$$I = \mathrm{O}(s \log n + s\,L),$$

the total length of a schedule (n jobs) may be exponential in I

(see also Cosmadakis and Papadimitriou).

Further work (1)

- McCormick, Smallwood and Spieksma, *Math. OR* (2001): multiprocessor scheduling with small number of p_j*'s*
- Agnetis, *Annals of OR* (1997): no-wait flow-shop
- Clifford and Posner, *Operations Research* (2000), *Math. Programming* (2001)

Further work (2)

- Grigoriev, Ph.D. Thesis, Maastricht, 2003
- Brauner, Crama, Grigoriev and Van de Klundert, *Journal of Combinatorial Optimization* (2005), *Statistica Neerlandica* (2007).
- Brauner and Crama, *Discrete Applied Mathematics* (2004)
- Grigoriev and Van de Klundert, *Discrete Optimization* (2006)

On the complexity of HMSP

Brauner, Crama, Grigoriev and Van de Klundert (2005, 2007)

Motivation:

• refine some of the crude complexity analysis found in Clifford and Posner *Math. Prog.* (2001)

• draw parallel with complexity analysis of list generating algorithms (Johnson, Yannakakis and Papadimitriou *Inf. Proc. Letters* 1988).

List-generating algorithms

Basic question:

- How should we analyze the complexity of an algorithm which is required to output a list of objects whose size is exponential in the size of the input??

List-generating algorithms

Examples:

- Generate all vertices of a polyhedron given by a system of linear inequalities.
- Generate all maximal stable sets of a graph.
- Generate all Pareto-optimal (efficient) solutions of a multicriteria optimization problem.

We can say that such problems are NP-hard... but it's not really fair!

List-generating algorithms

Main point is:

- Input size = I
- Output size = M
- M is exponential in I

Then, we call an algorithm *total polynomial* if its total running time is polynomial in I and M.

The algo runs with *polynomial delay* if the running time between successive outputs is polynomial in I (total time is $O(I\, M)$).

List-generating algorithms

Johnson, Yannakakis and Papadimitriou *Inf. Proc. Letters* (1988) for stable sets in graphs,

Fukuda (1996) for vertices of polyhedra,

T'Kindt, Bouibede-Hocine and Esswein (2005) for multicriteria scheduling problems,

Boros, Elbassioni, Gurvich, Khachiyan, Makino for other classes of problems,

etc.

Back to HMSP...

Since the number of jobs n is exponential in the input size I, distinguish among algorithms which

- compute the optimal schedule length in polynomial time *poly(I)* (compact encoding);
- list all starting times in total polynomial time *poly(n)* ;
- list all starting times with polynomial delay *poly(I)* between job k and job $k+1$;
- compute the starting time of job k in pointwise polynomial time *poly(I)*, for any k.

Example: 1-machine batch scheduling

Input: number n of identical jobs, processing time p, batch setup time b (3 numbers).

Problem: Group jobs into batches so as to minimize the sum of completion times.

The number of batches may be large ($\sqrt{n}$), but Shallcross (1992) computes the optimal value in polynomial time and can compute the size of the k-th batch in polynomial time for any k.

Example: Flowshops with flexible operations

- 2-machine flowshop, buffer of size b
- n identical parts
- Fixed operations can only be processed on a specific machine: total processing time of the fixed operations on M_1 is f_1, on M_2 is f_2 .
- One flexible operation can be processed on either machine; processing time s.
- Input size is

$$I = O(\log(b) + \log(n) + \log(f_1) + \log(f_2) + \log(s))$$

Example: Flowshops with flexible operations

- Input size is

$$I = O(\log(b) + \log(n) + \log(f_1) + \log(f_2) + \log(s))$$

- A solution consists of an assignment of the flexible operation to one of the machines for each part, and of a production schedule.
- Writing down a solution requires $O(n)$ time and space.

Problem is investigated in Crama and Gultekin *Journal of Scheduling* (2010).

Example: Flowshops with flexible operations

Crama and Gultekin (2010): when b is either 0 or infinite, pointwise polynomial algorithms

- require $O(I)$ computing time to determine the optimal makespan, and
- require $O(I)$ computing time to determine the starting time of any given part.

Example: Flowshops with flexible operations

Crama and Gultekin (2010): when b is positive and finite, polynomial-delay algorithm

- proceeds sequentially, part after part;
- requires $O(I)$ computing time to determine the assignment of the flexible operation for the next part;
- requires $O(I)$ computing time to determine the optimal makespan.

Open: Is there a pointwise polynomial algorithm for this problem?

Just in Time sequencing

- s product types;
- n_i items of type i ($i = 1,\ldots,s$);
- unit processing times.

Let $r_i = n_i / n$, where n = total number of jobs.

Determine a sequence of items such that, at every time k, the number of items of type i which have been processed is as close as possible to $k\, r_i$.

Just in Time sequencing: example

$n_1 = 3$ $n_2 = 3$ $n_3 = 1$

$r_1 = 3/7$ $r_2 = 3/7$ $r_3 = 1/7$

kr_1	3/7	6/7	9/7	12/7	15/7	18/7	21/7
x_{1k}	1	1	2	2	2	3	3
dev	4/7	1/7	5/7	2/7	1/7	3/7	0

JIT sequencing: total deviation

Different versions of the problem.

Let x_{ik} = number of items of type i processed up to time k ($i = 1,\ldots,s$; $k = 1,\ldots,n$).

Kubiak and Sethi, *Management Science* (1991):

$$\text{minimize} \sum_i \sum_k f(x_{ik} - k\, r_i)$$

where $f(.) = |\,.\,|$ or $(.)^2$ or …

Solvable in time $O(n^3)$: pseudo-polynomial (Kubiak *EJOR* 1993).

JIT sequencing: maximum deviation

Steiner and Yeomans, *Manag. Science* (1993):

(MDJIT) minimize $\max_{i,k} | x_{ik} - k\, r_i |$

Thresholding approach: fix maximum allowed deviation B.

We want to produce the j-th item of type i at time k so that $| j - k\, r_i | \leq B$.

MDJIT : earliest and latest dates

We want to produce the j-th item of type i at time k so that $| j - k\, r_i | \leq B$.

Bounds on k can be computed:

- earliest due date for j-th item of type i is

$$E(i,j) = \lceil (j - B) / r_i \rceil;$$

- latest due date is

$$L(i,j) = \lfloor (j - 1 + B) / r_i + 1 \rfloor.$$

MDJIT: Bipartite matching

Reduction to bipartite matching: graph *G*

- V = { product items } ∪ { time units}
- *j*-th item of type *i* is linked to all time units in the feasible interval [E(*i,j*) , L(*i,j*)].

<u>Proposition</u> (SY93): MDJIT has a solution with value at most *B* if and only if *G* has a perfect matching.

MDJIT : EDD algorithm

Since G is convex, the existence of a perfect matching can be checked in time $O(n)$ by the Earliest Due Date algorithm (Glover 1967):

- run through time periods $k = 1,\ldots,n$;
- assign to k the item (i,j) with earliest due date, i.e., with smallest value of $L(i,j)$ among all available items.

MDJIT : pseudo-polynomial algo

Binary search on B leads to O(n log n) algorithm for the optimization problem: pseudo-polynomial.

- Can we do better ?
- Is the MDJIT problem in P ? in NP ?

MDJIT: further results
(Brauner and Crama *DAM* 2004)

Idea:

- use Hall's theorem for the existence of a bipartite perfect matching :

  for all $X \subseteq \{\text{items}\}$, $| X | \leq | N(X) |$;

- specialize for convex graphs ;
- express in algebraic form.

This leads to:

MDJIT: algebraic characterization

Theorem:

MDJIT has a solution with maximum deviation at most B if and only if the following inequalities hold for all $x_1 \leq x_2$ in $\{1,2,\ldots,n\}$:

$$\sum_i \max\left(0, \lfloor x_2\, r_i + B \rfloor - \lceil (x_1 - 1)\, r_i + B \rceil\right) \geq x_2 - x_1 + 1$$

$$\sum_i \max\left(0, \lceil x_2\, r_i - B \rceil - \lfloor (x_1 - 1)\, r_i + B \rfloor\right) \leq x_2 - x_1 + 1.$$

MDJIT: co-NP and fixed s

Corollary 1: MDJIT is in *co-NP*.

Corollary 2: for fixed s, the optimal value of MDJIT can be solved in polynomial time.

Proof: express the CNS as linear inequalities in integer variables; use Lenstra's algorithm.

When $s = 2$, the problem is easy.

We don't know anything better when $s = 3$.

MDJIT: polynomial delay

Corollary 3: for fixed s, the optimal sequence can be determined with polynomial delay between job k and job $k+1$.

Proof: determine the optimal value B^* in polynomial time, then use the EDD algorithm.

MDJIT : optimal value

Corollary 4: the optimal value B^* of MDJIT satisfies :

$$B^* \leq 1 - 1/n.$$

Corollary 5: if $\gcd(n_1, n_2, \ldots, n_s) = m$, then the optimal solution is obtained by repeating m times the optimal solution for $(n_1/m, n_2/m, \ldots, n_s/m)$.

So, for MDJIT, it is not possible to reduce the average cycle time by duplicating the MPS.

MDJIT: small deviation instances

Note that $B^* < 1$ for all instances.

When is $B^* < 1/2$?

Conjecture: When $s \geq 3$, $B^* < 1/2$ if and only if $(n_1, n_2, \ldots, n_s) = (1, 2, 4, \ldots, 2^{s-1})$.

True for $s \leq 6$ (Brauner and Crama 2004).

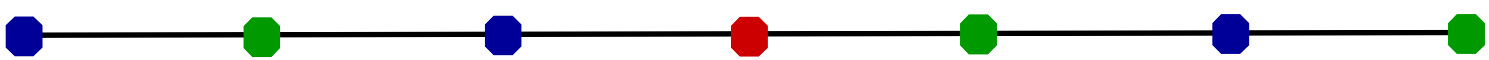
True for all s (Kubiak 2003; Brauner & Jost 2008).

MDJIT and Fraenkel's conjecture

Interesting connections with *balanced words* (« uniformly dense » colorings of integers) and *Fraenkel's conjecture* in number theory.

Balanced words

A *balanced word* is a coloring of the integers $\mathbb{N}$ with s colors such that, for any two subintervals $I1, I2$ of $\mathbb{N}$ of the same length, each color appears almost the same number of times in $I1$ and in $I2$ (« almost » means: up to one unit).



The *density* of color i in a balanced word is (roughly) the proportion of integers of that color in large intervals.

Fraenkel’s conjecture

Conjecture: When $s \geq 3$, there exists a balanced word on s colors with densities $(r_1, r_2, \ldots, r_s)$ if and only if $r_i \sim 2^{i-1}$.

The MDJIT conjecture is Fraenkel's conjecture for symmetric words.

Fair apportionment

Apportionment problem: Given s political parties and target ratios $(r_1, r_2, \ldots, r_s)$, allocate n seats in an assembly so that party i receives approximately $r_i n$ seats.

Closely related to JIT sequencing.

See: Kubiak, *Proportional Optimization and Fairness*, Springer 2009.

High multiplicity TSP

Description

- Graph $G = (V,E)$, $|V| = s$
- $s \times s$ distance matrix $D \geq 0$ (not necessarily symmetric, $d_{ii} \geq 0$)
- Integers n_i ($i = 1,2,\ldots,s$)
- Find the shortest tour which visits vertex i exactly n_i times, for $i = 1,2,\ldots,s$.

Example : Aircraft sequencing

(Psaraftis, *Operations Research* 1980)

- s categories of airplanes waiting to land (B747, B707, DC-9)
- there are several airplanes in each category; say, (5, 7, 3)
- landing duration and delay between successive landings depends on respective categories only.

High multiplicity TSP

- Model for machine scheduling with setups.
- Rothkopf (1966): conditions under which all jobs of a same type are processed in succession.
- Psaraftis (1980): dynamic programming pseudopolynomial algo: $O(\, s^2\, \Pi(n_i+1)\,)$.
- Cosmadakis and Papadimitriou (1984): $O(\, g(s)\, \log\, (\Sigma\, n_i)\,)$ where $g(s)$ is an exponential function of s; polynomial for fixed s.

Encodings of solutions (1)

Several possible encodings:

- sequence of vertices (jobs)
- solution (x_{ij}) of integer LP (x_{ij} = number of times edge (i,j) is traversed; transportation constraints + subtour elimination constraints)
- list (m_C, C) : m_C = number of copies of cycle C in the walk.

Encodings of solutions (2)

Size of different encodings:

- sequence of vertices : size = $\sum_i n_i$
- solution (x_{ij}) of integer LP : size = s^2
- list (m_C, C) : size = $O(s^2)$.

So, the HMTSP is in *NP*.

Non minimal part sets (1)

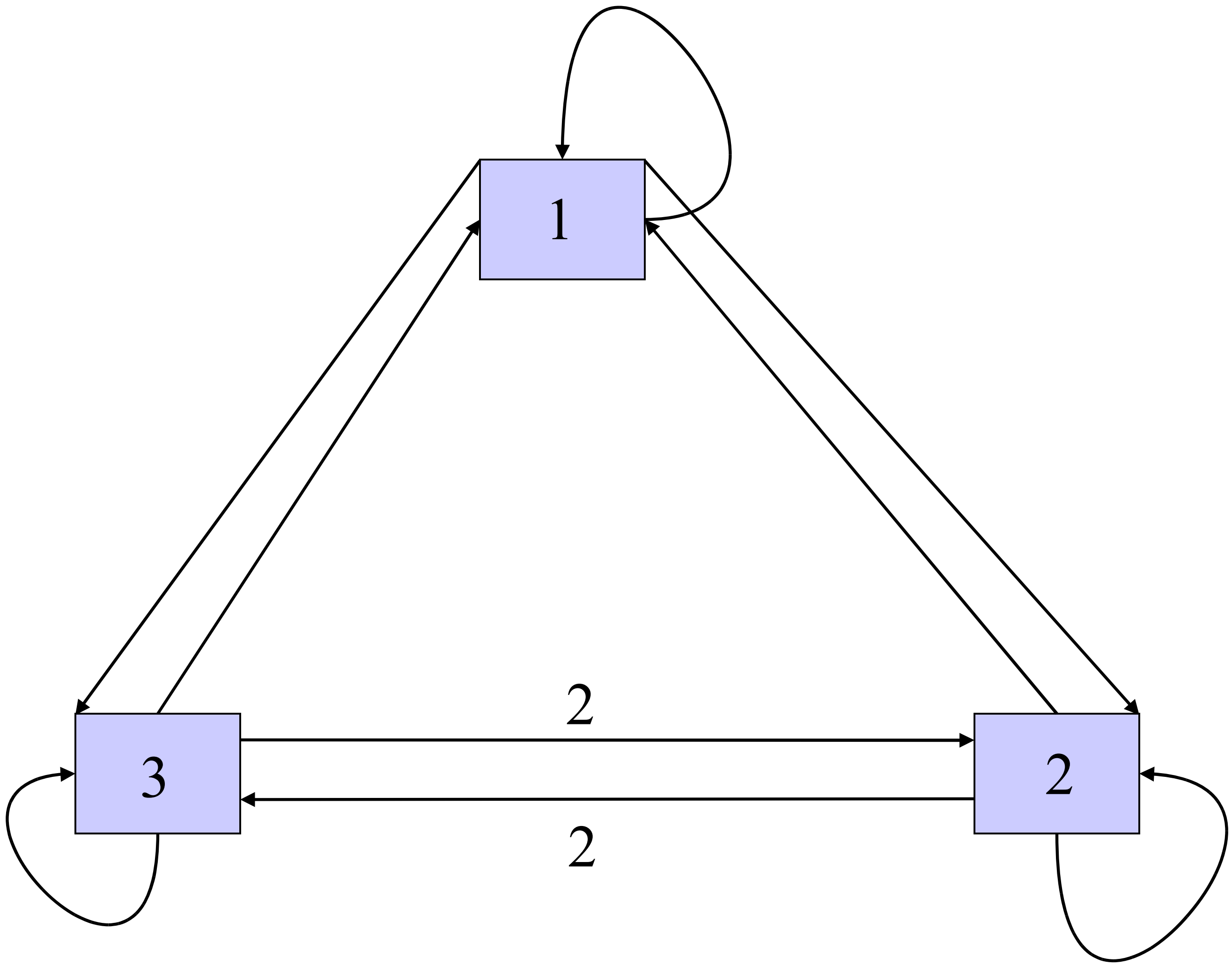
Back to Minimal Part Set (MPS):

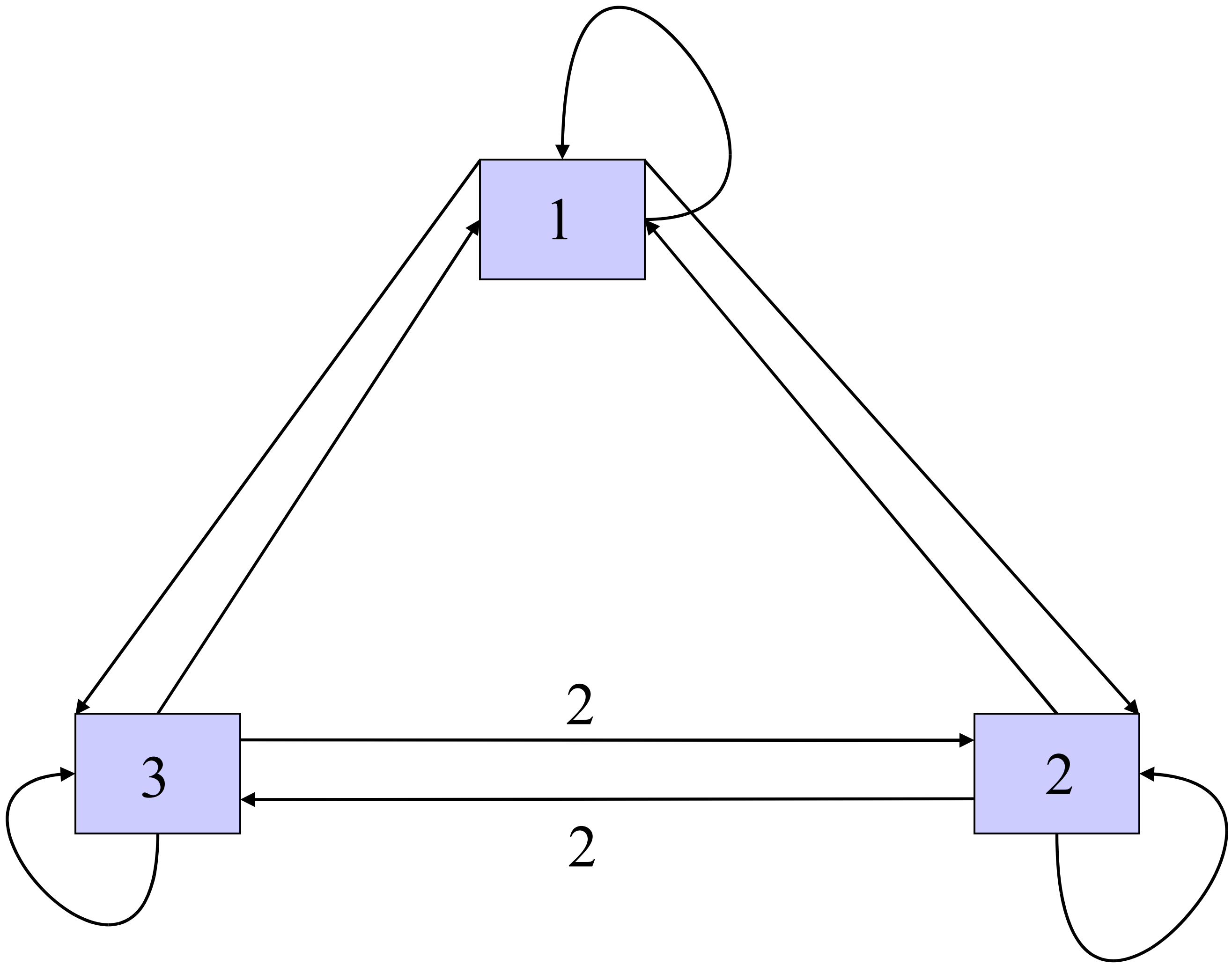
- production ratios are fixed: e.g., (1/2, 1/4, 1/8, 1/8)
- a Minimal Part Set (MPS) is a minimal batch of products which satisfies the target ratios and which can therefore be cyclically produced; e.g., MPS = (4, 2, 1, 1).
- Question: Is it possible to attain a smaller average cycle time if multiples of the MPS are produced cyclically ?

Non minimal part sets (2)

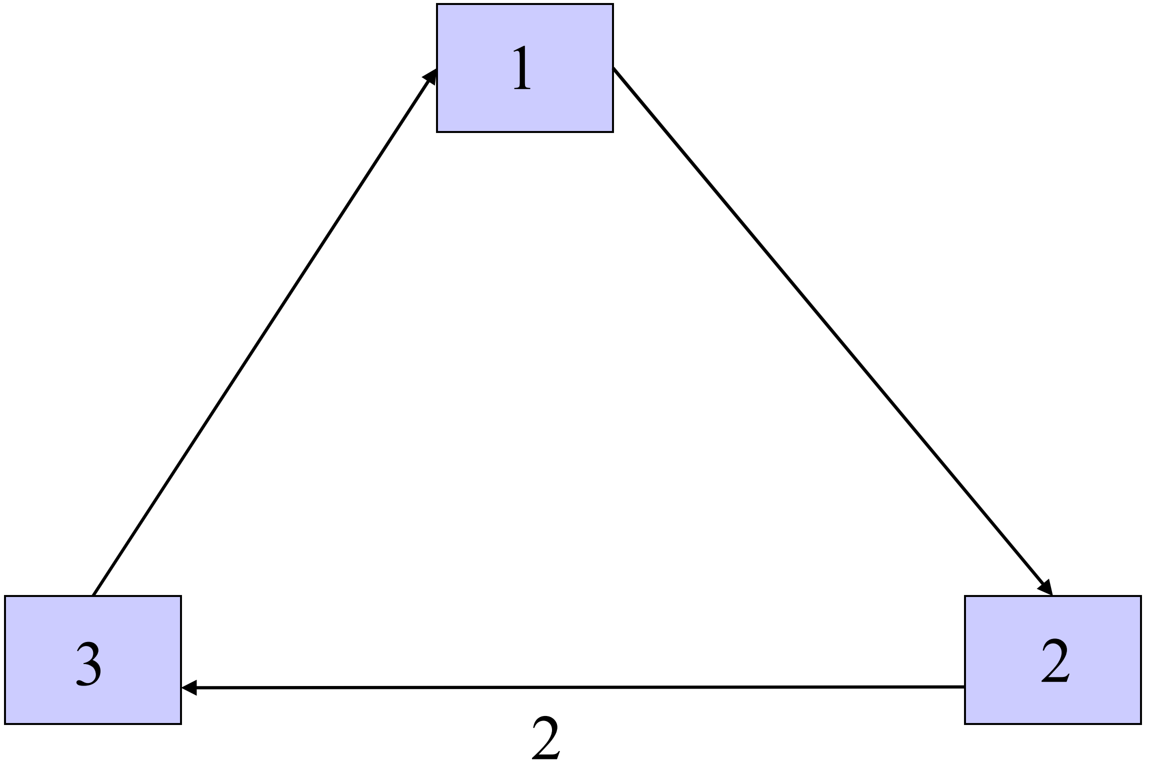
Is it possible to attain a smaller average cycle time if multiples of the MPS are produced cyclically ?

e.g., produce repeatedly (8, 4, 2, 2) instead of (4, 2, 1, 1).

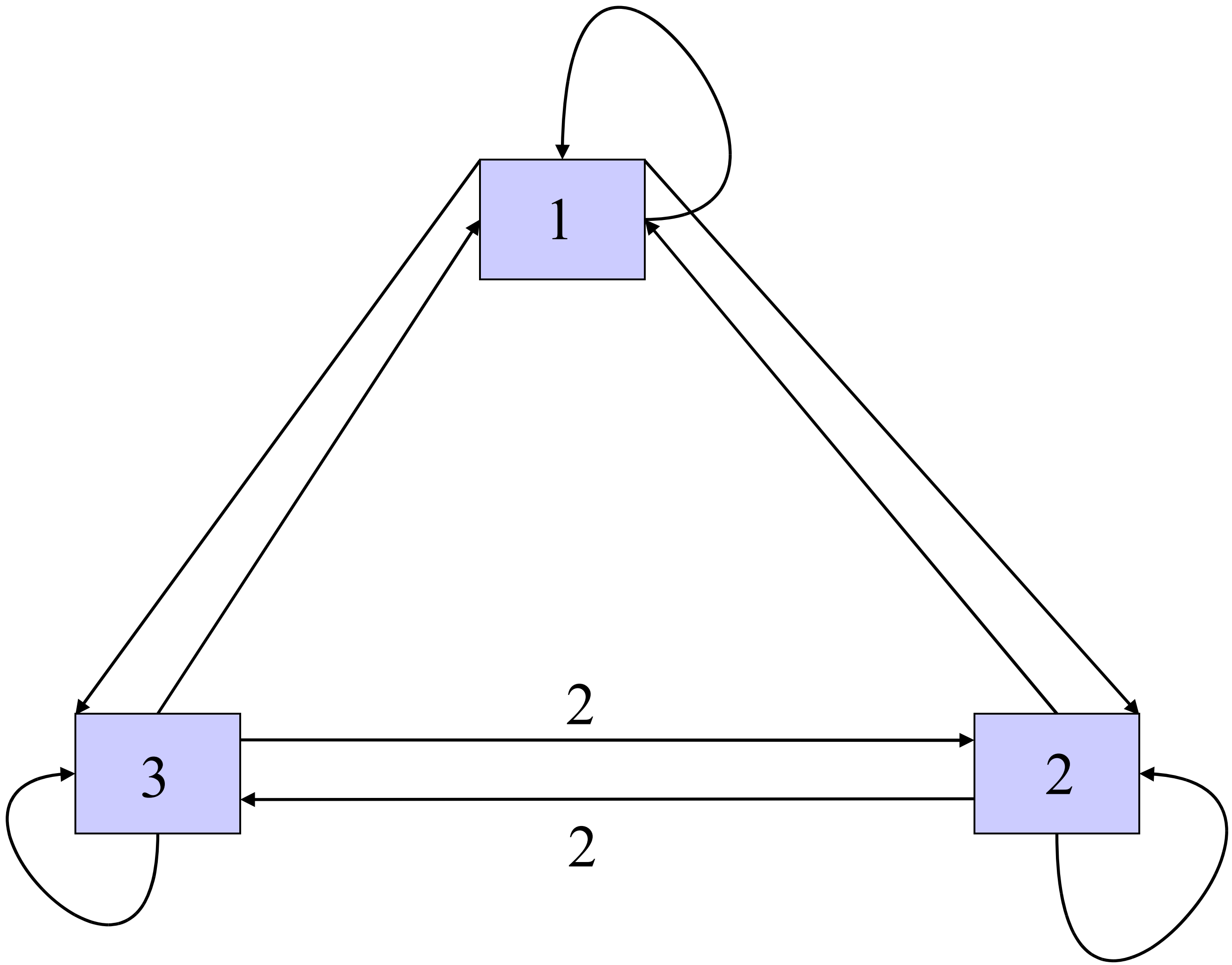

1
2
2
3
2



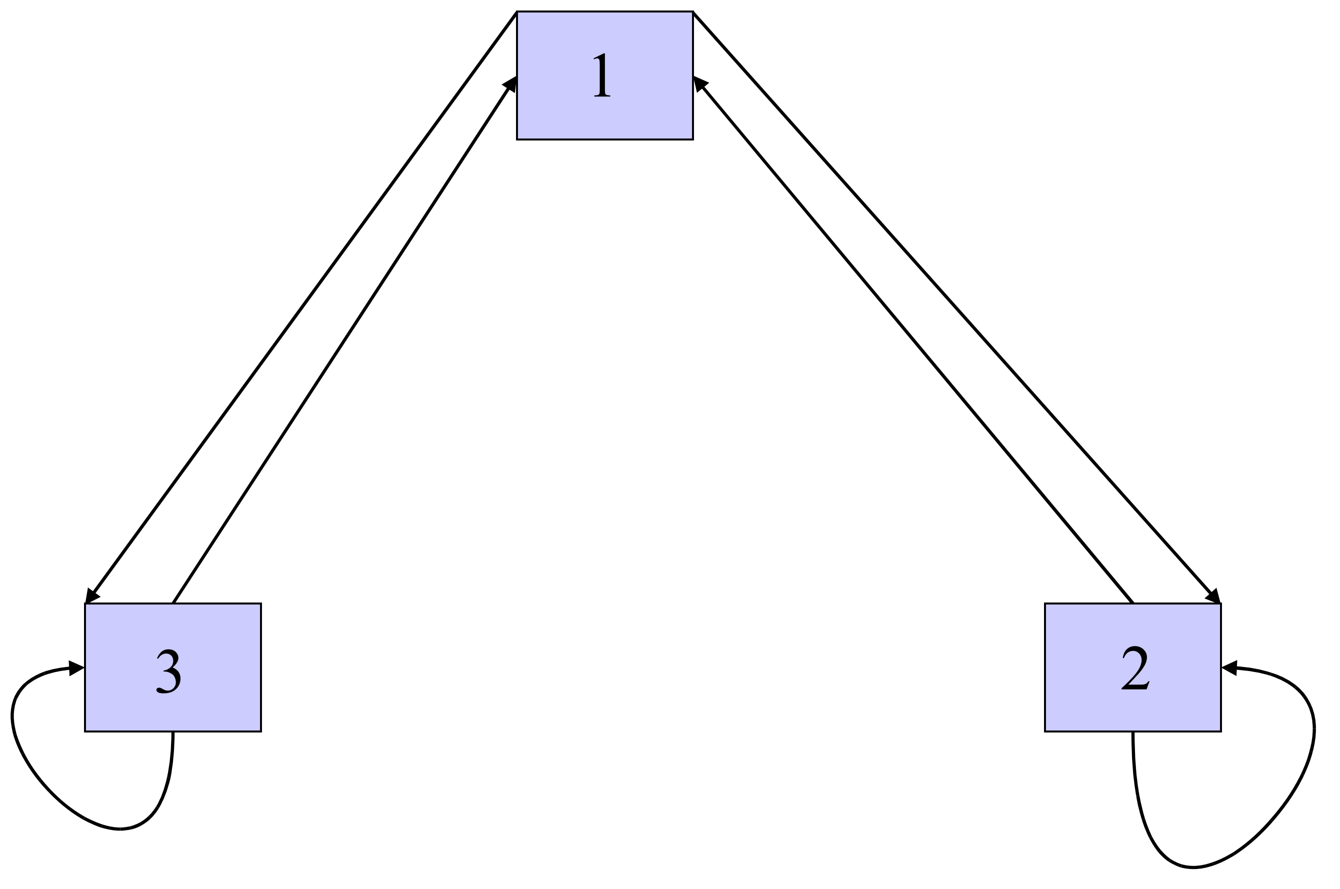
$$(n_1, n_2, n_3) = (1,1,1)$$



$(n_1, n_2, n_3) = (1,1,1)$ - Average tour length = 4



$$(n_1, n_2, n_3) = (2,2,2)$$



$(n_1, n_2, n_3) = (2,2,2)$ - Average tour length = 3

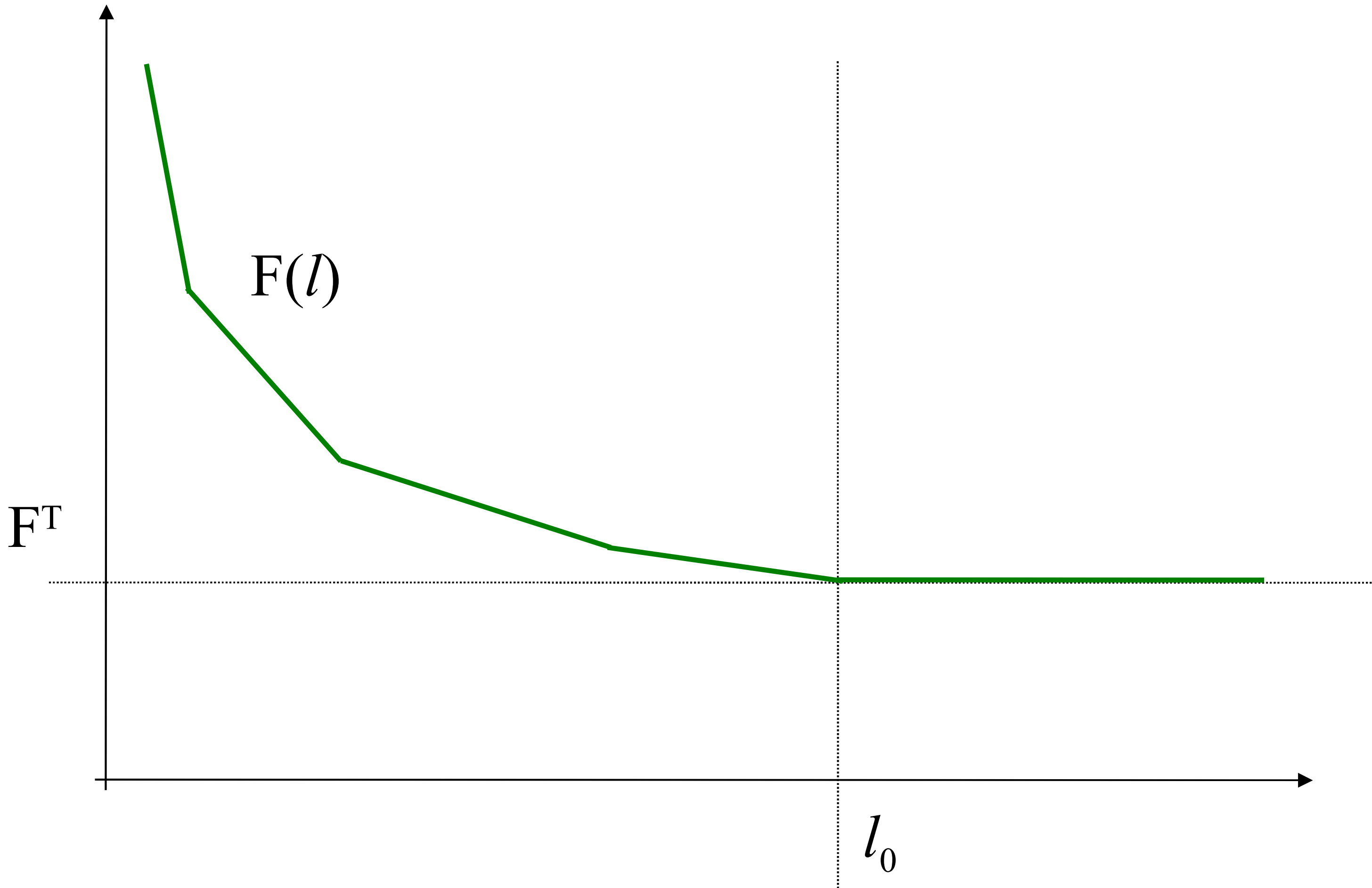
Results
(Grigoriev and Van de Klundert 2006)

Let $F(l)$: average tour length with $l \times n_i$ visits to city i ($i = 1,2,\ldots,s$).

Let F^T : optimal cost of a transportation problem with demands n_i and supplies n_j ($i, j = 1,2,\ldots,s$).

Theorem: for all $l \in \mathbb{N}$,

$$F^T \leq F(l+1) \leq F(l).$$


F(l)
F^T
l_0

Stable instances

An instance of HMTSP is stable if there exists l such that $F(l) = F^T$.

Let l^0 be the smallest such multiplier l.

Proposition. If l^0 exists, then $l^0 \leq s - 1$.

Proposition. Stable instances can be recognized in polynomial time.

Possible extensions ?

Basic question:

Is it possible to attain a smaller average cycle time if multiples of the MPS are produced cyclically ?

Remember: it is not the case for the MDJIT sequencing problem.

Other frameworks where this question could yield interesting results ?

Conclusions

- High multiplicity optimization problems pose intriguing and challenging complexity questions.
- Membership in P, NP, coNP may be non trivial.
- Algorithms can be viewed as list-generating algorithms.
- Connections with number theory and integer programming in fixed dimensions.
- Finding the optimal size of a part set (multipliers of the MPS) might be an interesting question in different settings.