Scheduling periodic messages and their answers on a single link

Maël Guiraud[1,2] and Yann Strozecki[1]

[1]David Laboratory, UVSQ
[2]Nokia Bell Labs France

***Abstract*—A recent trend in mobile networks is to centralize in distant data-centers processing units which, until now, were attached to antennas. The main challenge is to guarantee a low latency for the periodic messages sent from the antennas to their processing units and back to fulfill protocol time constraints. The problem we adress is to find a sending scheme from the antennas to their processing units and back without contention nor buffering.**

We focus on a simple but common star shaped topology, where all contentions are on a single arc shared by all antennas. For messages of arbitrary size, we show that there is always a solution as soon as the load of the network is less than 40%. Moreover, we explain how we can restrict our study to packet of size 1 without increasing too much the global latency.

For message of size 1, we prove that it is always possible to schedule them on a star shaped topology, when the load is less than $2/3$ using a polynomial time algorithm. Moreover, using a simple random greedy algorithm, we show that almost all instances of a given load admit a solution, explaining why most greedy algorithms work so well in practice.

I. Introduction

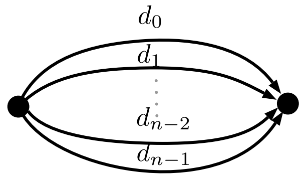
Lister les usages possibles: sans retour mais de profondeur 2 ou plus. Sonar ? Train ? Réseau de capteurs communicant (industrie) Logistique dans une usine (chaine avec différent produits)

Comparer à la litérature. Aller voir ce qu'on nous a cité sur l'ordonnancement périodique et les trucs de train périodique qu'on a trouvé. Dire que c'est du scheduling : contention point = machine et message = task.

Mélanger l'introduction et le modèle. Voir comment c'est présenté dans two flow shop

In this article, we model a very simple network in which periodic messages flow through a single link. The answer to each message is then sent back through the same bidirectional link. The model and problem can easily be generalized to any network, that is any directed acyclic graph with any number of contention points, see [1]. We choose here to present the simplest non trivial such network, for which we can still obtain some theoritical results.

The time is discretized and the process we consider is periodic of fixed integer period P. We use the notation $[P]$ for the set $\{0,\dots,P-1\}$. In our application, all messages are of the same nature, hence they are all of the same size denoted by τ. This size corresponds to the time it requires to send it through some contention point of the network. We denote by n the number of messages, which are numeroted from 0 to $n-1$. A message i is caracterized by its delay d_i. It means that if the message number i goes in the link at time t, then it returns to it in the other direction at a time $t + d_i$.



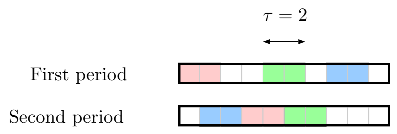
Since the process we describe is periodic, we chose any interval of P unit of time. Describing the messages going through the two contention points during such an interval completely define the periodic process. We call the representation of the interval of time in the first contention point the *first period* and the *second period* for the second contention point.

An *offset* of a message is a choice of time at which it arrives at the first contention point. Let us consider a message i of offset o_i, it uses the interval of time $[i]_1 = \{(o_i+t) \mod P \mid 0 \leq t < \tau\}$ in the first period and $[i]_2 = \{(d_i + o_i + t) \mod P \mid 0 \leq t < \tau\}$ in the second period. We say that two messages i and j collide if either $[i]_1 \cap [j]_1 \neq \emptyset$ or $[i]_2 \cap [j]_2 \neq \emptyset$.

We want to send all messages, so that they are no collision in the common link. In other word, we look for a way to send the messages without using buffering and hence limiting the latency to the physical length of the link. An assignment is a choice of an offset for each message such that *no pair of message collide*. Formally, an *assignment* is a function from the messages, represented by their indices in $[n]$ to their offsets in $[P]$.

We call Periodic Message Assignment or PMA the problem we study in this article, which asks, given an instance of n messages, a period P and a size τ to find an assignment or to decide there is none.

i	0	1	2
d_i	3	1	4



The complexity of PMA is not known. However, we have proven that when parametrized by n the number of messages, the problem is FPT [2]. A slight generalization of PMA, with several contention points but each message only goes through two as in PMA is NP-hard [2]. If the the link is not bidirectional, that is there is a single contention point and each message goes through it twice, it is also NP-hard []. Hence, we conjecture that PMA is NP-hard.

To overcome the hardness, we study PMA in this article when the load of the system is small enough. The *load* is defined as the number of unit of time used in a period by messages in a contention point divided by the period. Hence the load is equal to $n\tau/P$. Our aim is to prove that for small load, there is *always* an assignment and that it can be found by a polynomial time algorithm.

II. Greedy Algorithms for Large Messages

In this section, we study the case of large messages. When modelizing real problems, it is relevant to have $\tau > 1$ when the transmission time of a single message is large with regard to its delay.

A partial assignment A is a function defined from a subset S of $[n]$ to $[P]$. We say that $|S|$, the cardinal of the domain of A, is its *size*. We say that a message in S is *scheduled* (by A), and a message not in S is *unscheduled*. We only consider partial assignments such that no pair of messages of S collide. If A has domain S, and $i \notin S$, we define the extension of A to the message i by the offset o, denoted by $A[i \to o]$, the function defined as A on S and $A[i \to o](i) = o$.

All presented algorithms build the assignment incrementally, by growing the size of the domain of a partial assignment. Moreover, the algorithm of this section are greedy since once an offset is chosen for a message, it is never changed.

A. First fit

Assume that for some partial assignment A, the message i has offset o: it uses all times from o to $o + \tau - 1$ in the first period. If a message j is scheduled with some offset o' before o, then the last time it uses in the first period is $o' + \tau - 1$ and it should be less than o, which implies that $o' \leq o - \tau + 1$. If o' is larger than o, to avoid collision between messages j and i, it should be larger or equal to $o + \tau$. Hence the message i forbids $2\tau - 1$ positions for messages still not scheduled because of tis use of time in the first period. The same reasonning can be done for the second period, wich again forbids $2\tau - 1$ offsets. Hence, if $|S|$ messages are already scheduled, then $|S|(4\tau - 2)$ offsets are forbidden for the an unscheduled messages. Note that this is an upper bound on the number of forbidden offsets, since the same offset can be forbidden because of a message on the first and also on the second period.

We define $Fo(A)$, the maximum number of forbidden offsets when extending A. Formally, assume A is defined over S and $i \notin S$, it is the maximum over all values of d_i of $|\{ o \in [P] \mid A[i \to o]$ has no collision $\}|$. The previous paragraph shows that $Fo(A)$ is always bounded by $(4\tau - 2)|S|$.

The first algorithm deals with the route in the order they are given: for each unscheduled route it tests all offsets from 0 to $P - 1$ until one do not create a collision with the current assignment. We call this algorithm *First Fit*. Remark that if $Fo(A) < P$, then whatever the delay of the route we want to extend A with, it is possible to find an offset. Since $Fo(A) \leq (4\tau - 2)|S|$ and $|S| < n$, First Fit (or any greedy algorithm) will always succed when $(4\tau - 2)n \leq P$, that is when $n\tau/P \leq 1/4$. It turns out that First Fit always create compact assignments (as defined in [2]), that is a message is always next to another one in one of the two period. Hence, we can prove a better bound on $Fo(A)$, when A is built by First Fit, which implies the following therem.

Theorem 1. *First Fit always solves* PMA *positively on instances of load less than* $1/3$.

Proof. To prove the theorem, e prove that by induction on the size of S that $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$. For $S = 1$, it is clear since a single message forbid at most $(3\tau - 1) + \tau - 1 = 4\tau - 2$ offsets, as explained before. Now, assume $Fo(A) \leq |S|(3\tau - 1) + \tau - 1$ and consider a route $i \notin S$ such that First Fit builds $A[i \to o]$ from A. By definition of First Fit, choosing $o - 1$ as offset creates a collision, w.l.o.g. say it is a collision in the first period. It means that there is a shceduled message between $o - \tau$ and $o - 1$ (modulo P), hence all these offsets are forbidden by A. The same offsets are also forbidden by the choice of o as offset for i, then only $3\tau - 1$ new offsets are forbidden, that is $Fo(A[i \to o]) \leq Fo(A) + (3\tau - 1)$, which proves the induction and the theorem. ☐

B. Meta-intervals

The second method is described in [2] and achieves the same bound on the load using a different method which is also be used in the more involved algorithms of the next section. The idea is to restrict the possible offsets at which messages can be scheduled. It seems counter-intuitive, since it decreases artificially the number of possible offsets to schedule new messages. However, it allows to reduce the number of forbidden offsets, when the algorithm is designed accordingly. A *meta-offset* is an offset of value $i\tau$, with i an integer from 0 to P/τ. We call Meta-Offset the greedy algorithm which works as First Fit, but consider only meta-offsets when scheduling new messages.

We define $Fmo(A)$ as the maximal number of meta-offsets forbidden by A. By definition, two messages with a different meta-offset cannot collide in the first period. Hence, $Fmo(A)$ can be bounded by $3|S|$ and we obtain the following theorem.

Theorem 2 (Proposition 3 of [2])**.** *Meta-Offset always solves* PMA *positively on instances of load less than* $1/3$*.*

C. Tuples and meta-intervals

We now propose a more involved family of greedy algorithms which solves PMA positively for larger loads. We try to combine the good properties of the two previous algorithms: the compacity of the solutions of the first one and the use of meta-offsets ofthe second algorithm to reduce collisions on the first period. The idea is to schedule several messages at once, using meta-offsets, to maximize the compacity of the obtained solution.

We first describe the algorithm which places pairs of routes and then explain quickly how we can extend it by placing tuples of messages instead of pairs.

We first needs a lemma which allows to assume that the period P is a multiple of τ, which makes the analysis of our algorithms much simpler and tighter. It only changes the load from $n\tau/P$ to at most $n(\tau+1)/P$: the difference is less than $n/P < 1/\tau$, and is very small for large τ.

Lemma 3. *Let I be an instance of* PMA *with n messages of size τ and period P. Let $m = P/\tau$. There is an instance I' with n messages of size τ' and period $P' = m\tau'$ such that an assignment of I' can be transformed into an assignment of I.*

Proof. Let $P = m\tau + r$ with $r \leq \tau$. We define the instance I' as follows: $P' = mP$, $d_i' = md_i$ and $\tau' = m\tau + r$. With this choice, we have $P' = m(m\tau + r) = m\tau'$. Consider an assignment A' of the instance I'. If we let $\tau'' = m\tau$, then A' is also an assignment for $I'' = (P',\tau'',(d_0',\dots,d_{n-1}'))$ since we only reduce the size of each message (the interval of time used in the first and second period begi at the same position but are shorter). We then use a compactification procedure as in [2]. The first message is positionned at offset zero. The first time it uses in the second period is a multiple of m since its delay is by definition a multiple of m. Then all the other messages are translated to the left until there is a collision (remove increasing values from all offsets but the offset of the first message). It guarantees that some message is in contact with the first one, which implies that its offset is a multiple of m. This procedure can be repeated until we get some solution A'' to I'', and by induction it is proven that all positions of messages in the first and second period are multiple of m. Hence, if we define A as $A(i) = A''(i)/m$, we obtain an assignment of I. □

From now on, we always assume that $P = m\tau$ and the load is then n/m. We are interested in the remainder modulo τ of the delays of each message. We write $d_i = d_i'\tau + r_i$ and assume from now on that messages are sorted by increasing r_i. A *compact pair* is a pair of messages (i,j) with $i < j$ such that we can put them next to each other in the second period using meta-offsets. We have $r_i \leq r_j$ since $i < j$. We denote by g the gap between the two messages in the first period, that we define by $d_i' = g + 1 + d_j' \mod P/\tau$. We require that $g \neq 0$ so that there are no collision in the first period if we schedule these two messages such that the difference of their offsets is the gap.

Faire une dessin représentant le gap, la différence des restes et la paire compacte.

Lemma 4. *Given 3 messages, two of them always form a compact pair.*

Proof. If the first two messages or the first and the third message form a compact pair, then we are done. If not, then by definition $d_1' = 1 + d_2' = 1 + d_3'$. Hence, the messages 2 and 3 have the same delay and form a compact pair of gap 1. □

We call *Compact Pairs* the following greedy algorithm. From the n routes in order of increasing r_i, we build a sequence of at least $n/3$ compact pairs using Lemma 4. They are then scheduled in the order they have been built using meta-offsets only. If at some point all compact pair are scheduled or the current one cannot be scheduled, the remaining messages are scheduled as in Meta-Offset. The analysis of the algorithm relies on the evaluation the number of forbidden meta-offsets. In the first phase of the algorithm, one should evaluate the number of forbidden offsets when scheduling a compact pair, that we denote by $Fmo_2(A)$. In the second phase, we need to evaluate $Fmo(A)$. A compact pair only forbids three offsets in the second period instead of four if the two messages are scheduled independently, which explains the improvement from Meta Offset. We first give a simple lemma justified by Figure **??**, which allows to bound this number.

Lemma 5. *A compact pair already scheduled by Compact Pair forbid at most four meta offsets in the second period to another compact pair when it is scheduled by Compact Pair.*

Theorem 6. *Compact Pairs always solves* PMA *positively on instances of load less than* $3/8$*.*

Proof. Let us denote by n_2 the number of compact pairs scheduled in the algorithm. If we want to schedule a new pair, the position of the $2n_2$ messages on the first period forbid $4n_2$ offsets for a compact pair. Indeed, each scheduled message can collide with each of the two messages which form a compact pair. On the second period, we can use Lemma 5 to bound the number of forbiden offsets by $4n_2$. Hence, we have established that at the end of the first phase, the partial solution A satisfies $Fmo_2(A) \leq 8n_2$. This first phase continue while there is possible offsets for compact pairs, which is the case when $Fmo_2(A) \leq m$, that is while $n_2 \leq m/8$. In the second phase, a compact pair forbids 3 meta offsets in the second period and 2 in the first. Hence, if we let n_1 be the number of messages scheduled in the second phase to give the assignment A, we have $Fmo(A) \leq n_2 * 5 + n_1 * 3$. The algorithm can always schedule messages when $Fmo(A)$ is less than m, thus

$$n_2 * 5 + n_1 * 3 \geq m$$
$$n_1 \geq \frac{8m - 5m}{24}$$
$$n_1 \geq \frac{1}{8}$$

The number of routes scheduled is thus at least $2n_2 + n_1$, which is $\frac{3}{8}m$. Remark that we need to be able to schedule two third of the messages as compact pairs, which is possible by Lemma 4. Therefore an assignment is always produced when the load is less then $\frac{3}{8}$. □

The algorithm can be improved by forming compact tuples instead of compact pairs. A compact k-tuple is a sequence of messages $i_1 < \cdots < i_k$ (with $r_{i_1}, \ldots, r_{i_k}$) increasing, for which meta offsets can be chosen so that there is no collision and the messages in the second period are in the order $i_1, \ldots, i_k$, two consecutive messages i, j have offset o and $o + r_j - r_i + \tau$.

Dessin d'une paire compacte et d'untriplet compact et de comment ils se gênent.

The algorithms *Compact k-tuples* works by scheduling compact k-uples using meta offsets while possible, then scheduling compact $k-1$-uples and so on until $k = 1$. The theorem we give is obtained for $k = 10$, but taking arbitrary large k and using more refined bounds on $Fmo(A)$ do not even yields a load better than $41/100$ and only works for larger n.

Theorem 7. *Compact* 10*-uples always solves* PMA *positively on instances of load less than* $4/10$*, for instance with* n *large enough.*

Proof. We use two facts, which generalizes Lemma 4 and 5 to prove this theorem:

- Given $k^3/3$ messages, one can build from them a compact k-tuple. The compact k-uple is built greedily using pigeon-hole principle or the fact that if there are k messages with the same delay, they form a compact k-uple.
- A k-tuples forbid $k + j + 1$ offsets in the second period when scheduling a j-tuple. If the remainder of the messages in the j-tuples are larger than in the k-tuples, it forbids $k + j$ messages only.

The first fact garantees there are compact k-tuples, if there are enough messages, hence n should be larger than $k^3/3$. The second one allows to estimate how many i-tuple for i equal k down to 1 can be scheduled by bounding $Fmo_i(A)$, the number of forbiden meta-offsets when placing i-tuple in the algorithm. If we denote by n_i the number of compact i-tuples scheduled by the algorithm, we have the following equation:

$$Fmo_j(A) \leq \sum_{j=i}^{k} n_j(j * i + j + i + 1).$$

The equation for n_1 is slightly better:

$$Fmo(A) \leq \sum_{j=1}^{k} n_j(2j + 1).$$

A bound on n_i can be computed, since A can be extended while $Fmo_j(A) < m$. A numerical computation of the n_i's shows that the algorithm always find a solution when the load is less than $4/10$. □

D. Experimental results

Results better in practice, the worst case is different from the average case. The algorithms to test:

- first fit
- choix randomisé de l'offset possible
- meta offset
- heuristic to build super compact assignment: among the compact assignments possible, chose the one which minimizes the number of forbidden offsets
- algo avec les paires tant que c'est possible, si on a le temps
- FPT pour avoir la borne absolue

On représente le taux de succès en fonction de la charge. On peut faire pour des paquets de taille 1000 et une période de 100000 comme ça on peut faire une charge par pas de 1 pour cent. On peut sans doute commencer le graphique à 33 pour cent car la plupart des algos sont garantis marcher tout le temps en dessous. Meilleure idée ?

Quality of the results partially explained by the average analysis done later.

III. Reduction from large message to message of size one

In this section, we explain how we can restrict ourselves to the study of small τ and even $\tau = 1$ if we are willing to accept some buffering in our original problem or if we increase the load.

A. Reduction without buffering

We give a reduction from an instance of PMA to another one: from $I = (P, \tau, (d_0, \ldots, d_{n-1}))$, we build $I' = P, 2\tau, (d_0 - (d_0 \mod 2\tau) + \tau), \ldots, d_{n-1} - (d_{n-1} \mod 2\tau) + \tau))$. The instance I' has a load twice as large as I. On the other hand, all its delays are multiples of 2τ hence it is equivalent to the instance $I'' = P/2\tau, 1, d_0/2\tau, \ldots, d_{n-1}/2\tau))$. An assignment A' of I'can be transformed into an assignment A of I. Indeed, the message i uses the offset $A'(i)$ to $A'(i) + 2\tau - 1$ in the first period and $A'(i) + d_i - (d_i \mod 2\tau) + \tau$ to $A'(i) + 2\tau - 1 + d_i -$

$(d_i \mod 2\tau)+\tau$. The absolute value $\tau-(d_i \mod 2\tau)$ is less than τ by definition. If it is positive, then $A(i)=A'(i)$, if it is negative, then $A(i)=\tau+A'(i)$. By construction, a message scheduled by A uses a strict subset of the times in the first and second period used by the same message scheduled by A', which implies that A is an assignment of I if A' is an assignment of I'.

Ici un dessin pour expliquer la technique

B. Reduction using buffering

We have presented our problem with a single degree of freedom by message: its offset. We could also allow to buffer a message i during a time b between the two contention points, which means changing d_i to d_i+b. The quality of the obtained solution is worst since the buffering add latency to the messages. We now describe how we can make a tradeoff between the added latency and the size of the messages, knwowing that smaller messages helps to schedule instances with higher load.

All messages are buffered enough time so that their d_i have the same remainder modulo τ. It costs at most τ of buffering, which is not so good for random instances (see [2] for algorithms optimizing the latency), but much better than the theoritical upper bound of P which allows to always find an assignment. We can choose the message with the longes route as reference remainder, hence that message needs zero buffering. However, the message with the second longest route may have a remainder of $\tau-1$, thus the worst case increase of total latency is $\tau-1$. When all delays are changed so that d_i is a multiple of τ, we have an easy reduction to the case of $\tau=1$, by dividing all values by τ.

We can do the same kind of transformation by buffering all messages, so that d_i is a multiple of τ/k. The cost in term of latency is then at most τ/k but the reduction yields messages of size k. For small size of messages, it is easy to get better algorithm for PMA, in particular for $\tau=1$ as we show in the next section.

Theorem 8. *Compact Pairs on instances with $\tau=2$ always solves* PMA *positively on instances of load less than $4/9$.*

Proof. We assume w.l.o.g that there are less message with even d_i than odd d_i. We schedule compact pairs of messages with even d_i, then we schedule single message with even d_i. The worst case is when there is the same number of the two types of messages. The number of forbidden offsets for n scheduled messages is bounded by

$$n/2(1+3/2)+n/2(1+1).$$

Hence, we can always schedule messages when $n\leq(4/9)m$. □

If we do the previous reduction to the case $\tau=1$ while optimizing for the average added latency, we win a factor of two. The only degree of freedom in the reduction is the choice of the reference remainder (all other delays are modified to have the same remainder). Define the average buffer time for a choice t of reference by $B(t)$. If we sum $B(t)$ for $t=0$ to $\tau-1$, the contribution of each message will be $\sum_{i=0}^{\tau-1} i$. Since there are n messages, the sum of the $B(t)$ for all t is $n\tau(\tau-1)/2$. There is at least one term of the sum less than the average, hence there is a t_0 such that $B(t_0)\leq n(\tau-1)/2$. In other word, the average delay for a message, with this choice of reference is less than $\tau/2$.

IV. Going Above Load of 1/2 for messages of size one

When $\tau=1$, *any greedy algorithm* finds a solution to PMA when the load is less than $1/2$ since $Fo(A)\leq(4\tau-2)|S|=2|S|$ where S is the number of scheduled messages. We now give a method which always finds a solution for load of $1/2+\epsilon$.

To go above $1/2$ of load, we use a two-pass algorithm. In the second pass, a greedy algorithm is used to schedule (or reschedule) a subsets of the messages, selected because they have less conflicts with the already scheduled messages. The first pass is not greedy, since we allow to change a scheduled message, but we could use a greedy first pass or even a single pass greedy algorithm to go over $1/2$. However, the proofs are much harder to write and the ϵ for which they hold is much smaller.

Definition 1. *The potential of a message of delay d, for a partial assignment A is the number of integers $i\in[P]$ such that i is used in the first period and $i+d \mod P$ is used in the second period. We denote by$Pot(A)$ the sum of potentials of all messages in the instance for A.*

We are counting farovable configurations for the messages in terms of forbiden offsets. Indeed, when i is used in the first period and $i+d \mod P$ is used in the second period, then the same offset is forbidden *twice* for a message of delay d.

Definition 2. *The potential of a position i, for a partial assignment A, is the number of messages of delay d such that $i+d \mod P$ is used by a route scheduled by A.*

By construction, the potentials of the positions satisfy the following simple invariant.

Lemma 9. *The sum of potentials of all positions for a partial with k scheduled messages is nk.*

We then link $Pot(A)$ to the potential of the positions in the first period.

Lemma 10. *The sum of potentials of all positions in the first period used by routes scheduled by A is equal to $Pot(A)$.*

We now describe the algorithm to solve our problem with load $1/2+\epsilon$. The first pass schedules messages so that the $Pot(A)$ increases fast enough whih the number of scheduled messages. At the end of this step, it can be shown that the potential of the obtained partial solution is larger than some value. Then, we select R the set of the ϵP messages of largest potential. The messages in R and in the partial solution are removed, then, using any greedy algorithm, the unscheduled messages not in R and the messages in R are added to the partial solution.

A. Swap and potential improvement

Let A be some partial solution of size k and let i be an unscheduled message of delay d. Assume that i cannot be used to extend A. The swap operation is the following: select a free position p in the first period, remove the message which uses the position $p+d$ in the second period from A and add i to A with offset p. We denote this operation by $Swap(i,p,A)$.

Lemma 11. *Let A be some partial solution of size k and let i be an unscheduled message. If i cannot be used to extend S, then either $Pot(A) \geq kn/2$ or there is a p such that $Pot(Swap(i,p,A)) > Pot(A)$.*

Proof. The positions in the first period can be partitionned into two part: P_u the positions used by some scheduled message and P_f the positions unused. Let us denote by V_f the sum of the potential of the positions in P_f and by V_u the sum of the potentials of the positions of P_u. By Lemma 10, since P_f and P_u partitions the positions, we have $V_f + V_u = kn$. Moreover, by Lemma 10, $Pot(A) = V_u$, then $V_f + Pot(A) = kn$.

By hypothesis, since i cannot be scheduled, for all $p \in P_f$, $p+d_i$ is used in the second period. We now define the function F which associates to $p \in P_f$ the position $A(j)$ such that there is a scheduled route j which uses $p+d$ in the second period, that is $A(j) + d_j = p + d \mod P$. The function F is an injection from P_f to P_u. Remark now that, if we compare $Swap(i,p,A)$ to A, on the second period the positions are used. Hence, the potential of each position stay the same after the swap. As a consequence, doing the operation $Swap(i,p,A)$ add to $Pot(A)$ the potential of the position p and removes the potential of the position $F(p)$. Assume now, to prove our lemma, that for all p, $Pot(Swap(i,p,A)) \leq Pot(A)$. It implies that for all p, the potential of p is smaller than the potential of $F(p)$. Hence, since F is an injection from P_f to P_u, we have that $V_f \leq V_u = Pot(A)$. Since $V_f + Pot(A) = kn$, we have that $Pot(A) \geq kn/2$. □

B. Analysis of the Algorithm

We give an analysis of the algorithm, showing that it works for some value of ϵ. To obtain a better ϵ, we will later show some refinments of this algorithm: a better selection of the values added in the second step, the possibility to repeat the first step to guarantee a higher potential.

Theorem 12. *The two steps algorithm solves positively our problem with load $1/2 + 1/16$.*

Proof. The first step of the algorithm guarantes that we obtain a partial solution A of size k such that $Pot(A) \geq kn/2$ by Lemma **??**. Moreover, we can assume $k \geq P/2$ since one can always schedule $P/2$ routes with any greedy algorithm.

At the end of the first step, we have a potential of at least $Pn/4$ and we select the ϵP routes of largest potential. We want them to be of potential at least $2\epsilon P, 2\epsilon P + 2, \dots, 4\epsilon P$.

En supposant ça, expliquer pour quoi l'algo marche. Puis régler le epsilon pour que ça marche.

Sort all routes by decreasing potential and assume that the previous condition is not met, that is the ith route in order of potential is of potential less than $4\epsilon P - 2i$. The potential of a single route is bounded by $P/2$ since each placed route contribute at most one to its potential. Therefore, the first i routes are of potential at most $P/2$ and the following ones of potential at most $4\epsilon P - 2i$. Therefore the potential is less than $iP/2 + (4\epsilon P - 2i)(n-i)$. This function is decreasing for $i \leq \epsilon P$, hence the potential should be less than $4\epsilon Pn$.

For the algorithm to succeed, we want the potential to be larger than $4\epsilon Pn$ so that the condition on the routes of largest potential is met. Hence we must satisfy the following equation:

$$Pn/4 \geq 4\epsilon Pn.$$

$$\epsilon \leq 1/16.$$

□

Pour améliorer les résultats on peut répéter l'algo de swap une fois qu'on a obtenu au moins $(1/2+\epsilon)P$ routes placées, pour obtenir $n^2/2$ en potentiel. On doit alors avoir $n^2/2 \geq 4\epsilon Pn$, ce qui donne $\epsilon \leq 1/14$. Si on veut pousser la technique plus loin, il faut améliorer la borne sur le potentiel. Au lieu de prendre les ϵP plus grandes routes, on prend pour i de 1 à ϵP la route de plus petit potentiel supérieur à $2\epsilon P$ et pour qu'elle existe, il suffit que le potentiel soit supérieur à $2\epsilon P(n-i)$. À vérifier, si à un moment tout est de potentiel $2\epsilon P$ au moins, alors c'est facile de conclure. On obtient alors $\epsilon \leq 1/6$, ce qui donne un algo dès que la charge est inférieure à $2/3$.

Autre possibilité, améliorer le potentiel en obtenant plus que la moyenne, en jouant notemment sur la symétrie Backward et Forward.

Expliquer les nombreuses raisons pourquoi ça marche mieux en pratique.

V. Algorithms for random instances

a) $\tau = 1$: We analyze the following process, called **Uniform Greedy** or UG. For each element in order, chose one admissible position uniformly at random. We analyze the probability that Uniform Greedy solve the problem, averaged over all possible instances. It turns out that this probability, for a fixed load strictly less than one goes to zero when m grows.

Définir l'ensemble des solutions de taille n parmi m.

Theorem 13. *Given an instance of size n uniformly at random UG produces a solution uniformly at random or fail.*

Proof. Regarder mes notes partielles pour compléter ça. $\square$

Let us denote by $P(m,n)$ the probability that UG fails at the $n_t h$ steps assuming it has not failed before.

Theorem 14. *We have*

$$P(m,n) = \frac{\binom{n}{2n-m}}{\binom{m}{n}}.$$

In particular, $P(m,n) \leq f(\lambda)^m$*, where* $f(\lambda) < 1$.

Proof. Probability independent of the shift of the n element, can say it is 0. It is the probability that two sets of size n in $[m]$ are of union $[m]$. It is the same as the probability that it contains a given set of size $m-n$. Could find an asymptotic online. $\square$

Can we make the same argument for a deterministic algorithm? The not average version of the argument is the previous proof.

Show that we can find a solution for load $1/2-\epsilon$, even for large τ if n is large enough, using the trnasformation without buffering.

VI. Experimental results for $\tau = 1$

VII. Lower bounds

Example/family of examples for which some greedy alg fail. Example/family of examples with a given load such that there are no feasible solution.

References

[1] B. Dominique, G. Maël, and S. Yann, "Deterministic scheduling of periodic messages for cloud ran," *arXiv preprint arXiv:1801.07029*, 2018.

[2] D. Barth, M. Guiraud, B. Leclerc, O. Marcé, and Y. Strozecki, "Deterministic scheduling of periodic messages for cloud ran," in *2018 25th International Conference on Telecommunications (ICT)*, pp. 405–410, IEEE, 2018.