Algorithmique et architectures parallèles
Partiel – correction

E. Agullo C. Tedeschi Y. Robert

January 3, 2007

1 Topologie en m-star

On trouve dans la littérature sous l'appelation *m-star graph* une topologie initialement développée comme une alternative à l'hypercube et définie de la façon suivante. Une m-star est un graphe symétrique $G = (V, E)$ tel que $|V| = m!$, chaque sommet représentant une permutation distincte de m éléments, et E l'ensemble des arcs tels que deux permutations (nœuds) distinctes sont connectées ssi on peut passer de l'une à l'autre en interchangeant son premier élément avec n'importe quel autre. Par exemple, dans une 3-star, le nœud labellé par 123 est relié aux nœuds 213 et 321.

Remarquons qu'à l'instar des hypercubes, on peut construire cette structure de façon récursive. En effet, une m-star est composée de m $(m-1)$-stars disjointes. La figure 1 est une ébauche de 4-star.

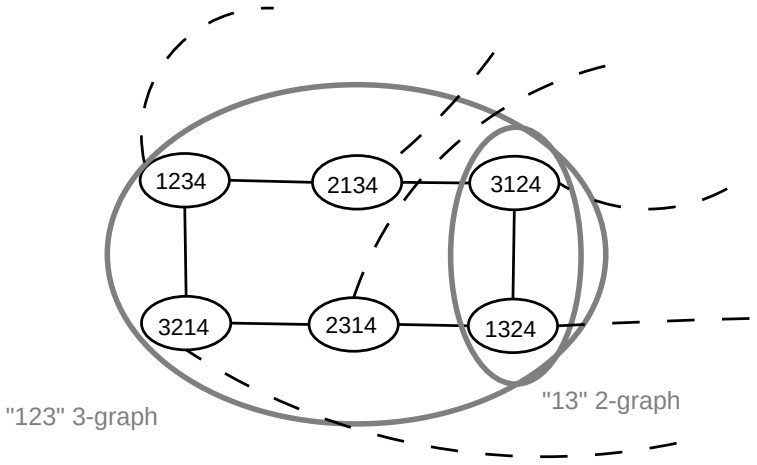

Figure 1: 4-star partielle

Question 1. *Dessinez une 4-star complète en expliquant votre méthode.*

La m-star est une structure récursive. Construire une m-star revient à relier m $m-1$-stars entre elles. Chaque $m-1$-star a ses somments labellés par les $(m-1)!$ permutations de l'une des m combinaisons de $m-1$ digits pris parmi m. Il faut ensuite ajouter à la fin des labels de chaque $m-1$-star le digit manquant. Enfin, la permutation du premier et du dernier élément de chaque label rajoutera les liens manquants (les autres liens étant déjà construits à l'intérieur de chaque $m-1$-star). La figure 2 illustre une telle construction pour $m = 4$.

Question 2. *Montrez qu'une m-star est un graphe $(m-1)$-régulier avec $|E_m| = m!\frac{m-1}{2}$.*

Par récurrence. Pour $m = 1$, notre graphe a un seul sommet de degré 0 (une seule permutation possible de 1 élément), donc on a bien un graphe 0-régulier avec $|E_0| = 0$. Supposons qu'une $m-1$-star soit un

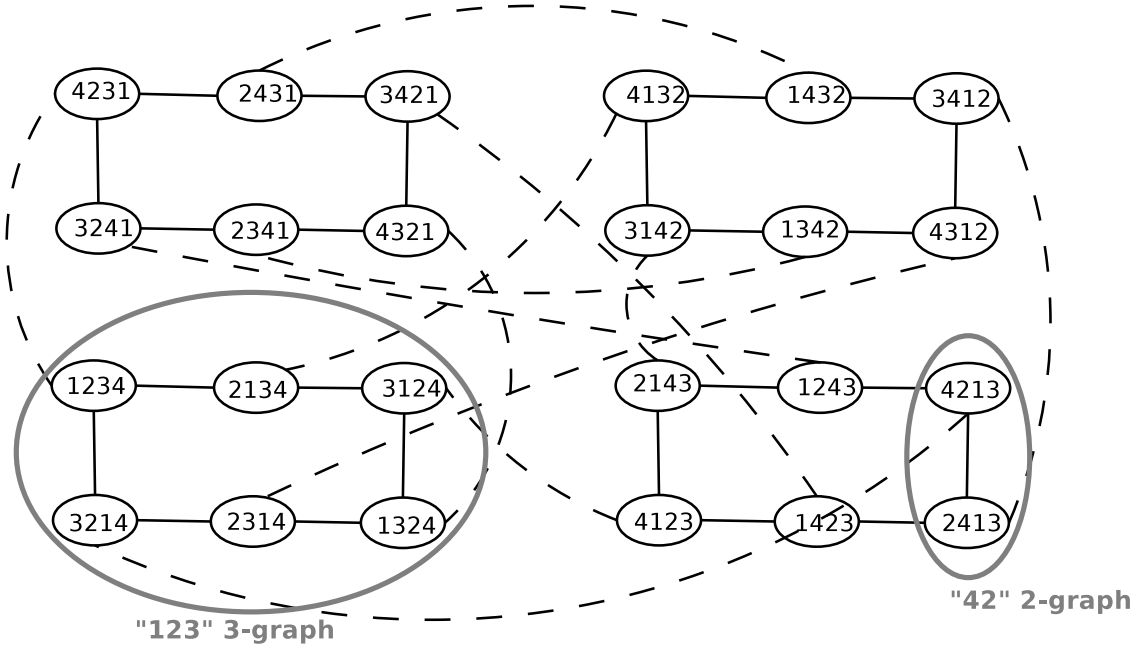

Figure 2: 4-star

graphe $(m-2)$-régulier avec $|E_{m-1}| = (m-1)!\frac{m-2}{2}, \forall m > 1$. Comme on l'a vu précédemment, une m star est composée de m $m-1$-stars auxquelles on rajoute un digit et des arêtes entre les permutations π_{1m} et pi_{m1}, ce qui ajoute un degré 1 sur chaque nœud, et on a donc un graphe $m-1$-régulier; et globalement $m/2$ arêtes:

$$|E_m| = m\frac{(m-1)!(m-2)}{2} + \frac{m!}{2} = \frac{m!}{2}(1+m-2) = m!\frac{m-1}{2}$$

Question 3. *Proposez un algorithme de routage d'un point à un autre dans une m-star et donnez sa complexité.*

On souhaite router de $A = a_1 \dots a_n$ à $B = b_1 b_n$ en mettant à jour A. On souhaite placer les éléments de chaque position en commençant par la fin, par exemple. Placer le i^e élément peut se faire en deux transitions. D'abord, on cherche j tq $a_j = b_i$ et on emprunte l'arête qui échange a_1 et a_j. Il nous reste à placer a_1 à la i^e position en empruntant l'arête qui échange a_1 et a_i. Ce qui nous donne l'algorithme suivant:

POUR $i = n$ to 2
  SI $a_i \neq b_i$
    TROUVER j tq $a_j = b_i$
    $A = a_j a_2 \dots a_{j-1} a_1 a_{j+1} \dots a_n$
    $A = a_i a_2 \dots a_{i-1} a_1 a_{i+1} \dots a_n$

La complexité au pire en nombre de sauts dans la topologie (tous les éléments doivent être changés) est de $2(m-1)-1$ (pour $n = 2$, la dernière opération fait du sur place).

Question 4. *Proposez un algorithme de diffusion depuis un nœud quelconque dans une m-star et donnez sa complexité.*

Une façon de diffuser est de propager l'information dans chaque $m-1$-star (en permutant le dernier digit, qui est différent pour chaque $m-1$-star avec tous les autres, en 2 étapes:

POUR $i = n - 1$ to 1
$A = a_i a_2 \dots a_{i-1} a_1 a_{i+1} \dots a_n$
$A = a_n a_2 \dots a_{i-1} a_1 a_{i+1} \dots a_i$

Puis de diffuser récursivement à l'intérieur de chaque $m-1$-star en parallèle. La procédure est appelée en parallèle pour m digits, chaque appel ayant une complexité de $2m$. La latence totale est donc en $O(m^2)$.

2 Plongement d'un anneau

Soit $G = (V, E)$ un réseau d'interconnexion arbitraire (mais connexe) à n sommets. On veut le configurer en anneau, i.e. plonger un cycle à n sommets dans le réseau. On utilise l'algorithme suivant:

- Construire un arbre couvrant de G.
- Partitionner les nœuds de l'arbre en sommets pairs (de niveau pair dans l'arbre) et impairs (de niveau impair dans l'arbre).
- Parcourir l'arbre en profondeur, ajouter un sommet pair à l'anneau si c'est sa première visite dans le parcours, et ajouter un sommet impair si c'est sa dernière visite dans le parcours.

Question 5. *Traitez un petit exemple.*
La figure 3 illustre un exemple de plongement. Les arêtes de l'anneau sont courbes et en gras et chaque somment est numéroté par son rang dans le parcours en profondeur (première visite).

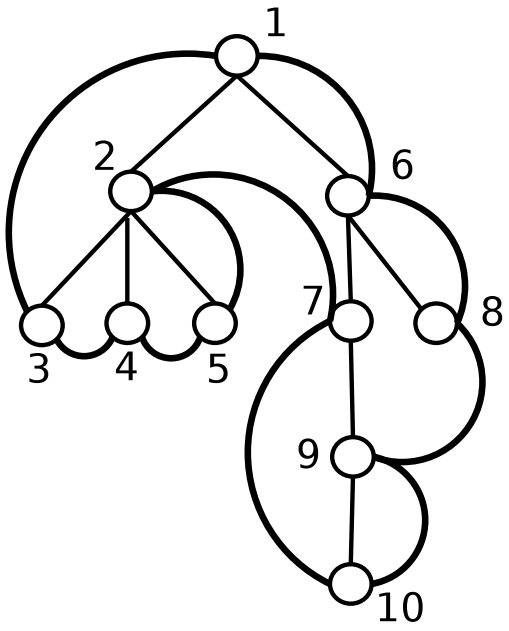

Figure 3: Plongement d'un anneau dans un arbre

Question 6. *Dilatation: quelle est la distance maximale dans le réseau de deux sommets consécutifs dans l'anneau?*
Dans le parcours en profondeur, supposons qu'un sommet s vienne d'être ajouté et procédons par disjonction des cas.

- s est pair:
  - Si on remonte deux fois, on prend au passage le père impair de s qui n'a pas d'autre fils**d=1**.
  - Si on remonte une fois et que l'on redescend, on prend un frère pair de s **d=2**.
  - Si on descend, on tombe soit sur un fils impair de s sans fils $\mathbf{d} = \mathbf{1}$, soit sur un petit-fils pair **d=2**.
- s est impair:
  - Si on remonte trois fois, on prend le grand-père impair de s qui n'a pas d'autre fils **d=2**
  - Si on remonte deux fois et que l'on redescend une fois, on prend l'oncle pair de s **d=3**.
  - si on remonte une fois et qu'on redescend deux fois, on prend un neveu pair de s **d=3**.
  - si on remonte une fois et qu'on redescend qu'une fois avant de remonter, on prend le frère impair de s qui n'a pas de fils **d=2**.
  - si on descend une fois, on prend un fils pair de s $\mathbf{d} = \mathbf{1}$.

On ne peut pas faire plus de 3 sauts sans rajouter un sommet, le diamètre du plongement est 3.

Question 7. *Partage: combien de fois un lien donné du réseau est-il partagé dans l'anneau?*

Dans le parcours en profondeur, on passe par chaque lien exactement deux fois. Chaque lien dans est partagé par deux liens dans l'anneau.

3 Ordonnancement sur m-star

Reprenons la topologie étudiée dans l'exercice 1 et faisons maintenant abstraction de la numérotation des nœuds. On considère une m-star comme un ensemble de processeurs numérotés de 0 à $m! - 1$. On peut découper une m-star en un nombre quelconque d'intervalles contigus, tant que les intervalles sont des i-stars valides, avec $1 \leq i \leq m$. On notera $[a, b]$ l'intervalle des processeurs contigus numérotés de a à b (avec $a < b$) formant une i-star valide. Pour un l quelconque, une m-star peut être divisée en l intervalles consécutifs $[a_1, b_1] \dots [a_l, b_l]$ où $a_1 = 0$ et $b_l = m! - 1$.

On souhaite maintenant ordonnancer n tâches indépendantes $J = (J_1, J_2, \dots, J_n)$ avec $J_i = (d_i, t_i)$ sur une m-star. Cela signifie que J_i doit s'exécuter sur une d_i-star (donc sur $d_i!$ processeurs) pendant t_i unités de temps. t_i est rationnel. On a toujours $1 \leq d_i \leq m$. On suppose de plus que ces tâches sont préemptives. Autrement dit, elles peuvent être arrêter au cours de leur exécution et reprendre plus tard là où elles s'étaient arrêtés, éventuellement sur un autre ensemble de processeurs, mais toujours de dimension d_i. On cherche à écrire un algorithme qui décide s'il est possible d'ordonnancer l'ensemble des tâches J de façon à ce que leur exécution soit finie avant une date butoir T.

Question 8. *Montrez d'abord que pour que l'ordonnancement soit valide, on doit avoir* $\forall i, 1 \leq i \leq n$: $t_i \leq T$ *et* $\frac{1}{m!} \sum_{i=1}^{n} t_i d_i! \leq T$.

Pour que l'ordonnancement soit valide, c'est-à-dire, il faut que chaque tâche ait une durée inférieure à la date butoir, c'est-à-dire $\forall i, 1 \leq i \leq n : t_i \leq T$. Le temps total de travail disponible pendant T est $m!T$. Le temps total requis est la somme des durée de chaque tâche par son nombre de processeurs, c'est-à-dire $\sum_{i=1}^{n} t_i d_i!$. Pour avoir un ordonnancement valide, il faut au préalable s'assurer que le travail requis est inférieur au travail disponible, c'est-à-dire $\sum_{i=1}^{n} t_i d_i! \leq m!T$.

On cherchera à ordonnancer les tâches (non ordonnées) une par une. On notera à chaque pas (après chaque tâche i ordonnancée) l'état de la plate-forme de la façon suivante:

$$S_{i-1} = ([a_1, b_1], r_1), ([a_2, b_2], r_2), \dots, ([a_k, b_k], r_k)$$

où $([a_j, b_j], r_j)$ signifie que pour la suite contiguë de processeurs $[a_j, b_j]$, le temps de calcul disponible restant est de r_j, autrement dit l'ordonnancement des $i - 1$ premières tâches necessite $T - r_j$ sur $[a_j, b_j]$.

L'état initial est $S_0 = ([0, m! - 1], T)$. On pensera à ne garder que les ensembles de processeurs pour lesquels $r \neq 0$ et à trier les $([a_j, b_j], r_j)$ par r_j croissant.

Question 9. *En utilisant les indications précédentes, écrivez un algorithme qui décide s'il existe un ordonnancement optimal pour l'ensemble J, c'est-à-dire se termiant au plus tard à l'instant T. Donnez une idée de la preuve de correction de votre algorithme et sa complexité.*

Question 10. *Illustrez votre algorithme sur l'exemple suivant: 5-star, $T = 4$, $J = (J_1, J_2, J_3, J_4, J_5, J_6)$ avec $J_1 = (4, 2)$, $J_2 = (4, 4)$, $J_3 = (4, 3)$, $J_4 = (3, 3)$, $J_5 = (3, 3.5)$ et $J_6 = (1, 4)$.*

Les réponses aux questions 9 et 10 se trouvent dans l'article suivant:

Shahram Latifi and Pradip K. Srimani. Preemptive Job Scheduling in Star Graph Networks. Technical Report CS-96-119 Colorado State University. 1996.