INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Message Scheduling for Data Redistribution through High Performance Networks

Frédéric Wagner — Emmanuel Jeannot

N° 5077

janvier 2004

THÈME 1

Rapport de recherche

N 0249-6399

Message Scheduling for Data Redistribution through High Performance Networks

Frédéric Wagner* , Emmanuel Jeannot

Thème 1 — Réseaux et systèmes
Projets Algorille

Rapport de recherche n° 5077 — janvier 2004 — 31 pages

Abstract: With the emergence of large scale distributed computing, new problems bound to data transfers are appearing. We present here the problem of data redistribution between two clusters connected by a high performance network. This problem consists in finding the best way to transfer data from the first cluster to the second one in the shortest possible time. In order to avoid slowing down the network, and the transfer, it is necessary to schedule the messages. This NP-complete problem (named as KBPS) has already been studied. We recall the model chosen, and study the advantages and drawbacks of the existing resolution methods. We prove that the existing heuristics are not approximation algorithms, and can in some case give some very bad results. We then develop two new polynomial-time approximation algorithms. The proof of the approximation factor and the complexity in time are presented in detail. To validate the theoretical work achieved, we conclude with results obtained from simulations and experiments on real clusters.

Key-words: data redistribution, messages scheduling, bipartite graphs, grid computing, kbps

* This work is partially supported by the Région Lorraine, the french ministry of research ACI GRID, ARC redGRID

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Ordonnancement de messages pour la redistribution de données à travers un réseau à haut débit

Résumé : Avec l'émergence du calcul distribué à grande échelle, de nombreux problèmes liés aux transferts de données apparaissent. Nous présentons ici le problème de la redistribution de données entre deux grappes d'ordinateurs reliées par un réseau à haut débit. Ce problème consiste à transférer des données de la première grappe vers la seconde, dans un temps le plus court possible. Afin de ne pas congestionner le réseau, et donc de ne pas ralentir le transfert, il est nécessaire d'ordonnancer les messages. Ce problème NP-complet (nommé KBPS) ayant déjà fait l'objet de travaux antérieurs, nous rappelons ici le modèle choisi, et nous étudions les avantages et inconvénients des méthodes de résolution existantes. Nous prouvons que les heuristiques existantes ne sont pas des algorithmes d'approximation, et peuvent donc dans certains cas donner de très mauvais résultats. Nous développons ensuite deux nouveaux algorithmes d'approximation polynomiaux. La preuve du facteur d'approximation ainsi que le calcul de la complexité en temps sont présentés en détail. Pour valider le travail théorique effectué, nous concluons par des résultats obtenus par des simulations ainsi que des tests sur des machines réelles.

Mots-clés : redistribution de données, ordonnancement de messages, graphes bipartis, grid, kbps

Part I
Problem

1. Introduction

With the emergence of grid computing many scientific applications use code coupling technologies to achieve their computations where parts of the code are distributed among parallel resources interconnected by a network. Code coupling requires data to be redistributed from one parallel machine to another. For instance the NxM ORNL project [12] has for objective to specify a parallel data redistribution interface and CUMULVS [9] (which uses MxN) supports interactive and remote visualization of images generated by a parallel computer. In this paper we concentrate on the scheduling of the messages when a parallel data redistribution has to be realized on a network, called a backbone. Two parallel machines are involved in the redistribution : the one that holds the data and the one that will receive the data. If the parallel redistribution pattern involves a lot of data transfers, the backbone can become a bottleneck. Thus, in order to minimize the parallel data redistribution time and to avoid the overloading of the backbone it is required to schedule each data transfer.

In this report, we revisit the problem of packet switching (in wavelength-division multiplexed (WDM) optical network [4, 8, 13, 15, 16] or in satellite-switched time division multiple access (SS/TDMA) [3, 10, 11]) in the context data redistribution.

Data redistribution has mainly been studied in the context of high performance parallel computing [1, 6, 7]. In this paper we study a generalization of the parallel data redistribution. Indeed, contrary to some previous works that were only dealing with block-cyclic redistribution [2, 7], here, no assumption is made on the redistribution pattern. Moreover, contrary to other works which assume that there is no bottleneck [1, 6], we suppose that the ratio between the throughput of the backbone and the throughput of each of the n nodes of the parallel machines is k. Hence, no more than k communications can take place at the same time. We study the problem for all values of k. We focus on the case $k < n$ (the backbone is a bottleneck) whereas the case $k \geq n$ has been tackled in [1, 6].

2. Model

In this report we shall consider data redistribution between two distant clusters connected by a backbone as shown in Figure 1. Each node each cluster is connected to the backbone by a switch. For each cluster we consider the bandwidth between a node and it's switch to be equal for each node. Each network card is assumed to be full-duplex and 1-port.

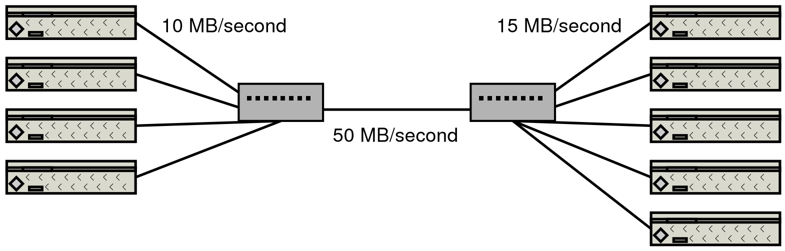

Figure 1. two distant clusters

Given the different bandwidths we can compute k, the maximum number of simultaneous connections generating no interferences. In the example of Figure 1, $k = 4$ since all nodes in the left cluster can communicate at full bandwidth without exceeding the backbone's bandwidth. If we note b_1, b_2 the bandwidths connecting cluster's nodes to switchs, bb the backbone bandwidth, v_1, v_2 the number of nodes in each cluster, we have $k = \min(v_1, v_2, \left\lfloor \frac{bb}{\min(b_1, b_2, bb)} \right\rfloor)$

The representation chosen to modelize our data redistribution problem is a weighted bipartite graph $G = (V_1, V_2, E, f)$, with V_1 representing the first cluster's nodes, V_2 the second cluster's nodes and $E \subset V_1 \times V_2$ (the graph's edges) the communications to perform. The f function ($f : E \rightarrow \mathbb{Q}$) represents the cost in terms of time associated to each communication.

For example consider the communications of Figure 2 (on the previously introduced network) in Megabytes :

cluster nodes	$y1$	$y2$	$y3$	$y4$	$y5$
$x1$	30	20	0	0	15
$x2$	20	0	20	0	0
$x3$	0	0	40	20	0
$x4$	10	10	10	10	0

Figure 2. Communication matrix

They are represented by the bipartite graph of Figure 3. To obtain this graph we first need to convert the megabytes to communicate into a time unit. Therefore we divide each entry in the communication matrix by the worst bandwidth in the network (min(b_1, b_2, bb)). In our example we have at least 10 Megabytes/second, so the 40 MB communication from node $x3$ to node $y3$ will take us $40/10 = 4$ units of time.

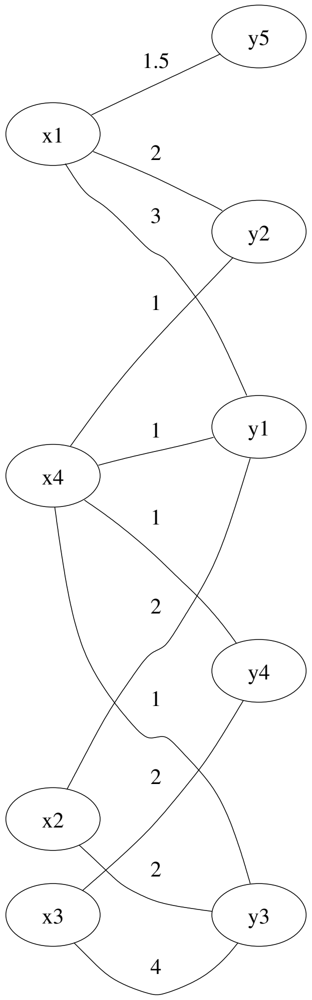

Figure 3. Communications of Figure 2 seen as a graph

3. KBPS

3.1. Presentation

We are looking for the best way to execute all communications. This means we want to use a maximal amount of bandwidth, and avoid losses due to a network saturation. Two constraints are given by the network :

- 1-port : since all cards are 1-port in order to avoid conflicts we need to ensure a node will not be emitting/receiving two communications at the same time.
- k : we also need to take into account the maximum number of simultaneous communications generating no interferences.

This maps on our model by making sure we do not choose two edges incidents at a same node, and choosing at max k edges to issue simultaneously. Which means we are looking for weighted matchings of at most k elements. We therefore divide the communications into a set of matchings, each of cardinality less than k. We consider preemption possible, so communications may be split into several smaller chunks. We will refer to KBPS (or K-Bipartite Scheduling Problem) as the problem of finding the best possible set of matchings, knowing the bipartite graph and k. KBPS is an extension of the Bipartite Scheduling Problem studied by [6] with the constraint given by k.

Consider for example the bipartite graph of Figure 4. A valid decomposition into three matchings is given in Figure 5. As we accept preemption, numerous other possibilities are possible.

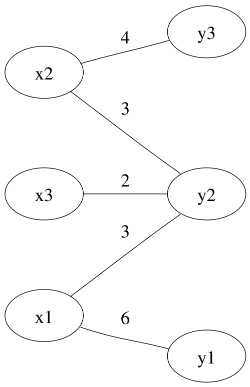

Figure 4. initial graph, k = 3

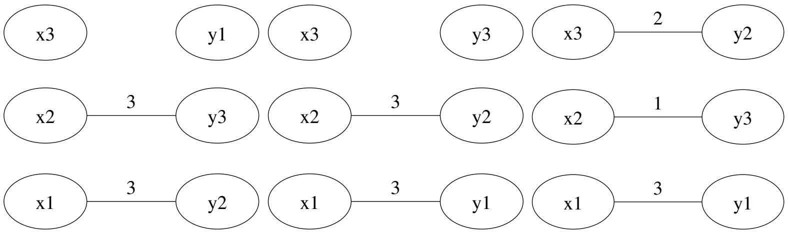

Figure 5. valid decomposition

3.2. Lower Bounds

We call $G = (V_1, V_2, E, f)$ our bipartite graph, with V_1 and V_2 the two sets of nodes, $E \subset V_1 \times V_2$ the edges, and $f : E \rightarrow \mathbb{Q}$ the weight function.

We note: $m(G) = |E(G)|$ the number of edges, $\Delta(G)$ the maximal degree of all nodes, $w(s)$ the sum of the weights of all edges incident to node s, $W(G)$ the max of all $w(s)$, $P(G)$ the sum of the weights of all edges, η_e the minimal number of communication steps, and η_d the minimal amount of time used to transfer datas.

We have :

- $\eta_e = \max\left(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil\right)$ since we cannot make two simultaneous communications to/from one node (hence the $\Delta(G)$) and we cannot make more than k communications at the same time (hence the $\left\lceil \frac{m(G)}{k} \right\rceil$).
- $\eta_d = \max\left(W(G), \left\lceil \frac{P(G)}{k} \right\rceil\right)$ for the same reasons as above.

With all these notations we can compute the optimal redistribution cost η as:

$$\eta = \eta_d + \beta\eta_e$$

In the example of Figure 4 we have : $m(G) = 5$, $k = 3$, $W(G) = 9$, $P(G) = 18$, $\Delta(G) = 3$ so :
$\eta = \eta_d + \beta\eta_e = \max\left(W(G), \left\lceil \frac{P(G)}{k} \right\rceil\right) + \beta\max\left(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil\right) = \max(9,6) + \beta\max(3,2) = 9 + 3\beta$.

The cost of the decomposition of Figure 5 is 3β (since we have three steps of communications) $+3 + 3 + 3$ (the maximal cost of each communication at each step) $= 9 + 3\beta$ which is the same as our optimal cost. We can therefore conclude that this particular solution of KBPS is optimal.

Note that in some cases the optimum may not be reachable. For example, in Figure 6 (with k = 2) the optimum is $\eta = 8 + 2\beta$ but no set of matchings may yield to that result.

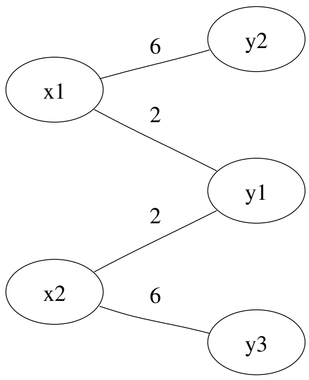

Figure 6. $\eta = 8 + 2\beta$

KBPS has been proven in [14] to be NP-complete. We shall therefore focus on heuristics and approximation algorithms to solve it.

Part II
Previous Work

Our work is an extension to the works of Cohen, Jeannot and Padoy, so we will present in detail two heuristics and one approximation algorithm [14] that we are going to use and modify.

4. Heuristics

4.1. Description

Cohen, Jeannot and Padoy propose two simples heuristics. They all take as input G, the bipartite weighted graph described in the previous part, k the maximal number of simultaneous connections, β the startup time cost of a connection. All edges weights are the time cost of data transfers. A common description of the heuristics is shown in Figure 7.

- Choose a maximal matching m in G
- Retain k edges in m matching a given criteria
- Remove the k selected edges from G
- Start again until G is empty

Figure 7. heuristics

The two heuristics behave in an identical manner. They differ only in the criteria used to select the edges. The two criterias we choosed are thought to decrease η_e and η_d. To decrease η_e we retain the k edges which are incidents to nodes of the highest degree, and to decrease η_d we choose the k edges of biggest weight.

Several tests on these heuristics are available in [14].

5. Approximation algorithm

5.0.1 Presentation

This algorithm has been developped by Cohen, Jeannot, and Padoy in [5]. Several ideas are guiding this algorithm. First, we should take advantage of preemption to optimize our scheduling.

Take for example the set of matchings of Figure 8.

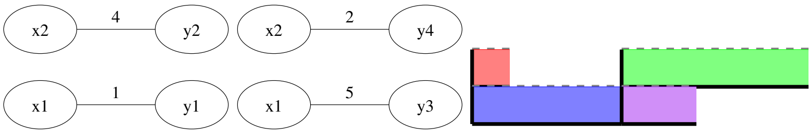

Figure 8. Set of matchings, corresponding chronogram

The chronogram is displaying how the communications effectively take place, in time. The "holes" we can see in the chronogram are showing us a case where we are not using the full available bandwidth. The optimal solution would be as shown in Figure 9 to split the communications into smaller ones, to achieve maximal throughput.

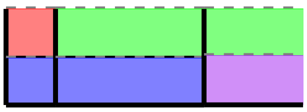

Figure 9. Better packing

Therefore one of our goals is to assure that in every communication step (i.e. in every matching) the transmission time is the same for all communications (weights are the same for every edge).

But, we should also take the startup time into account. Splitting communications may increase the number of steps needed. As each of these steps has a fixed cost (β) too much preemption may cost more than bring.

We will do the following: enable a communication to be preempted only if it's time is greater than β. To do that, we start by normalizing each edge's weight by β and rounding to the upper integer. We then split each edge of weight e into e edges by building a multigraph I. This way we will get the benefits of preemption.

We could then start computing the results but we first need to be sure that we will not select a matching with more than k edges.

We are going to add nodes and edges to our graph, to build a $\left\lceil \frac{m(I)}{k} \right\rceil$-regular graph. Each new edge will connect a new node with an existing node, ie no two new nodes will be connected. This operation is possible if and only if $\frac{m(I)}{k} \geq \Delta(I)$ and in case $\frac{m(I)}{k} < \Delta(I)$ we add some more edges until $\frac{m(I)}{k} = \Delta(I)$.
We are now building incrementaly the final set of matchings.

Property: In a regular graph, it is always possible to find a perfect matching.

So as the graph is now $\left\lceil \frac{m(I)}{k} \right\rceil$-regular, we are able to find a perfect matching. We add it to the set of results, and remove it from the graph. As the matching was perfect, the degree of each node decreases by one, and so the graph remains regular. We then start again until the graph is empty.

By using this method, we have as result $\left\lceil \frac{m(I)}{k} \right\rceil$ different matchings. Therefore we have (on average) $\frac{m(I)}{\left\lceil \frac{m(I)}{k} \right\rceil} \leq k$ "initial" edges per matching. In fact the number of "initial" edges per matching is always less than k, because the edges added to build a regular graph always saturate an "initial" node. For a more formal proof, you should refer to [14].

5.0.2 Algorithm

We consider $G = (V_1, V_2, E, f_G)$ the weighted bipartite graph representing our communications, k the number of simultaneous communications allowed, β the time cost for establishing a connection. We call R the set of matchings representing the solution of KBPS.

The algorithm is the following :

1. $R = \emptyset$
2. Build $H = (V_1, V_2, E, f_H)$ such that $\forall e \in E, f_H(e) = \left\lceil \frac{f_G(e)}{\beta} \right\rceil$
3. Build the multigraph $I = (V_1, V_2, E_I, f_I)$ such that $\forall e \in E_I, f_I(e) = 1$ and E_I is such that $\forall e \in E, e$ is $f_H(e)$ times in E_I
4. If $\frac{|E_I|}{k} < \Delta(I)$ build the multigraph $J = (V_{1_J}, V_{2_J}, E_J, f_J)$ such that :
   - $E \subset E_J, V_1 \subset V_{1_J}, V_2 \subset V_{2_J}$
   - $|V_{1_J}| - |V_1| = |V_{2_J}| - |V_2| = |E_J| - |E_I| = k\Delta(I) - |E_I|$
   - $\forall e \in E_J, f_J(e) = 1$
   - $\forall v \in V_{1_J} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_J} | v_2 \notin V_2$

   Else if $\frac{|E_I|}{k} \notin \mathbb{N}$
   - $E \subset E_J, V_1 \subset V_{1_J}, V_2 \subset V_{2_J}$
   - $|V_{1_J}| - |V_1| = |V_{2_J}| - |V_2| = |E_J| - |E_I| = k \left\lceil \frac{E_I}{k} \right\rceil - |E_I|$
   - $\forall e \in E_J, f_J(e) = 1$
   - $\forall v \in V_{1_J} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_J} | v_2 \notin V_2$

   Else $J = I$
5. Build the multigraph $K = (V_{1_K}, V_{2_K}, E_K, f_K)$ such that :
   - $E_J \subset E_K, V_{1_J} \subset V_{1_K}, V_{2_J} \subset V_{2_K}$
   - K is $\frac{m(I)}{k}$-regular
   - $\forall (v_1, v_2) \in E_K | (v_1, v_2) \notin E_J,$ either $v_1 \notin V_{1_J}$ or $v_2 \notin V_{2_J}$

- $\forall e \in E_K, f_K(e) = 1$

6. While $E_K \neq \emptyset$ do :
   (a) Choose a perfect matching M in K
   (b) Add M to S a set of matchings
   (c) Remove M from E_K
7. Remove all edges e in S such that $e \notin E$
8. While $S \neq \emptyset$
   (a) Choose a matching M in S
   (b) Modify f_M such that $\forall e \in M, f_M(e) = \min(\beta, f_G(e))$ and f_G such that $\forall e \in M, f_G(e) = f_G(e) - f_M(e)$
   (c) Add M to R

Example:
We are now going to see a small example. Consider the bipartite graph of Figure 10 with $k = 2, \beta = 2$.

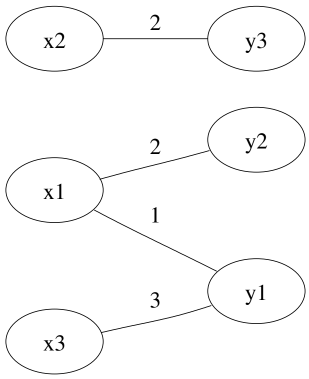

Figure 10. Initial graph, $\beta = 2, k = 2$

We first normalize each weight by 2, and round to the upper integer, building H. Then we transform H into the multigraph I as shown on Figure 11. We can now compute : $\Delta(I) = 3$, $|E(I)| = 5$, $\frac{|E(I)|}{k} = 2.5$, $\left\lceil \frac{|E(I)|}{k} \right\rceil = 3$.

We have $\frac{|E(I)|}{k} < \Delta(I)$, so to build J we add one edge (x_4, y_4). We can see we have $\frac{|E(J)|}{k} = \Delta(J) = 3$.

Now to build K we should add edges until K is 3-regular. So we need to add one edge on x_1 and x_3, and two edges on x_2 and x_4 for a total of 6 new edges, so we will need to add $6/3 = 2$ new nodes in V_2 The same for the edges in V_2: we need to add two edges on $y_2, y3, y_4$ and no edge on $y1$. This gives a total of 6 new edges, so we add $6/3 = 2$ nodes in V_1. In final K is the 3-regular graph displayed on Figure 12.

We can now start the main loop of the algorithm. We find a perfect matching (here dotted) as shown in Figure 12 and add it to S. Now we remove all edges of the matching from K and start again with the remaining graph of Figure 13. We choose a second perfect matching, add it to S and remove all of it's edges from the graph. The remaining graph is now 1-regular, so we take all of it's edges as the last matching.

The S set of matching contains 3 matchings of 6 edges each which are shown on Figure 14.

In order to obtain the final result, apply the final step of the algorithm, removing all non-initial edges and nodes, and changing weights back. The final set of 3 matchings is shown in Figure 15. The cost of our solution is : $2 + 2 + 1 + 3\beta = 11$ which isn't optimal, but within a factor two of the optimum (8).

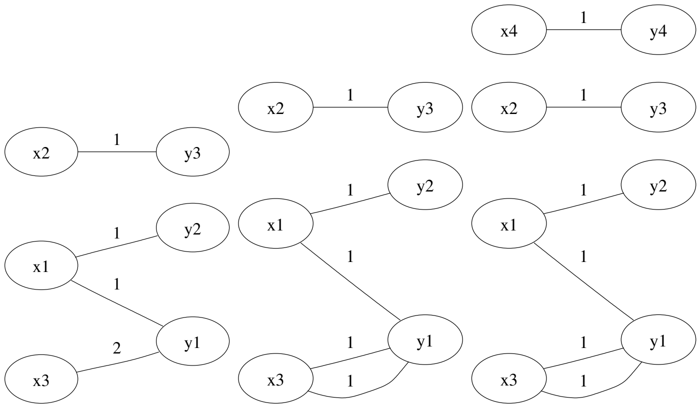

Figure 11. H, I, J

5.0.3 Properties

Complexity

In this algorithm, the most time-consuming part is the while loop we have at step 6. Finding a maximal matching using the hungarian method is in $O(\sqrt{n(G)} \times m(G))$. At each iteration we remove at least one edge, so we have at most $m(K)$ iterations. So, this algorithm is in $O(\sqrt{n(K)} \times m(K) \times m(K)) = O(\sqrt{n(K)} \times m(K)^2)$

But, to build K we need to turn each edge of E into $f_G(e)$ edges. This means that $m(K)$ is dependent from the weights of the edges. Therefore the algorithm is only pseudo-polynomial. A modified version of this algorithm exists in [14], with a polynomial complexity of at least $O(m(G)^2 n(G)^4)$.

Approximation factor

Since at each iteration in the main loop we take a perfect matching, we have in final $\eta_e(K)$ matchings. For each matching in S, communications will take at most 1 (startup time) $+1$ (communication time) $=$ 2 units of time. This means that the global time taken t will be at most $2\eta_e(K) = 2\max\left(\Delta(K), \left\lceil \frac{m(K)}{k} \right\rceil\right)$.

By construction, we have $\Delta(K) \leq W(I) + \Delta(I)$ and $m(K) \leq P(I) + m(I)$.

So we have :

$$t \leq 2\max\left(W(I) + \Delta(I), \left\lceil \frac{P(I) + m(I)}{k} \right\rceil\right)$$

$$\Rightarrow t \leq 2\max\left(W(I), \left\lceil \frac{P(I)}{k} \right\rceil\right) + 2\max\left(\Delta(I), \left\lceil \frac{m(I)}{k} \right\rceil\right)$$

$$\Rightarrow t \leq 2\eta_d(I) + 2\eta_e(I) = 2\eta(I)$$

At step 2, when normalizing by β we may only increase edge's weights. So we can conclude that the time cost of our scheduling is less or equal than $2\eta(H)$. This algorithm is a 2-approximation of KBPS.

Part III
Contributions

6. Heuristics

We have not developped new heuristics, but we made an in depth study of the properties they yield. We have been able to prove the following results:

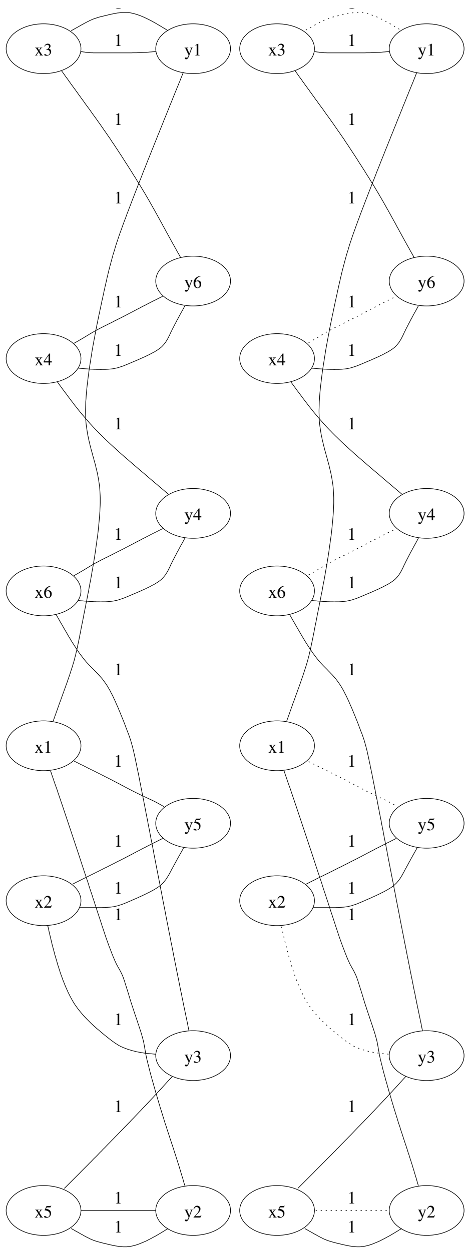

Figure 12. Building a 3-regular graph, finding first matching

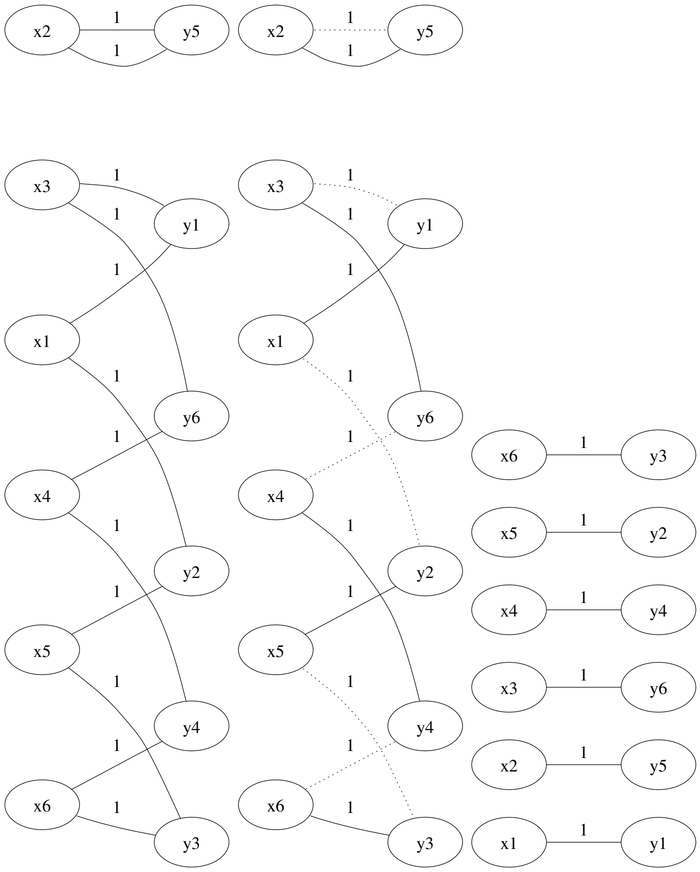

Figure 13. Removing matching, choosing another one

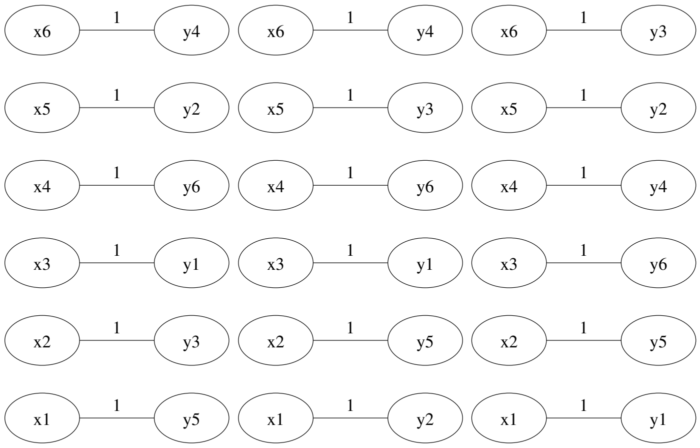

Figure 14. S

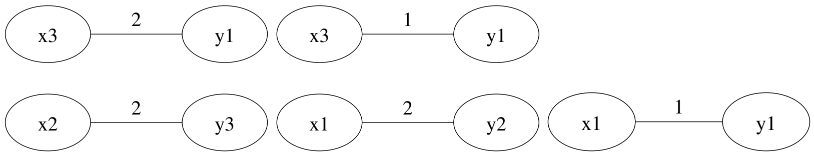

Figure 15. Final results

- the number of steps we obtain is at most $3\eta_e$.
- the approximation ratio may be as bad as k.

6.1. Number of steps

We shall now prove that the number of steps obtained using the heuristics is at worst $3\eta_e$.

Lemma:
G_r being a graph with the following property : G_r may be decomposed in a set S of maximal matchings each of cardinality $< k$. We have $|S| \leq 2\eta_e(G_r)$.

Proof by contradiction:
Suppose there exists a set S of at least $2\Delta(G_r) + 1$ maximal matchings. Let s be a node of degree $\Delta(G_r)$. s is in $\Delta(G_r)$ differents matchings. Then this means that s is not in $|S| - \Delta(G_r)$ matchings. Let $M \in S$ be one of such matchings which does not contain s. M is maximal therefore all neighbour nodes of s are in M. If that were not the case there would be a neighbour s_2 of s such that we could extend M by adding the edge (s, s_2) to him. As s is not in $|S| - \Delta(G_r)$ matchings, his neighbours are in at least $|S| - \Delta(G_r)$ matchings. Since $|S| \geq 2\Delta(G_r) + 1$ each of s neighbours are in at least $2\Delta(G_r) + 1 - \Delta(G_r) = \Delta(G_r) + 1$ matchings. This is not possible since they would need to have a degree higher than $\Delta(G_r)$.
We have a contradiction so we can conclude that in $G_r, |S| \leq 2\eta_e(G_r)$.

Theroem:
S being a set of maximal matchings (with cardinality $\leq k$) partitioning a bipartite graph G, we have $|S| < 3\eta_e(G)$.

Proof:
Let's be G the initial bipartite graph. We have a set S of maximal matchings partitioning G. We build G_r, the union of all matchings in S of cardinality $< k$, G_s the union of the remaining matchings, all of size k (S_s and S_r are the sets of matching building G_r and G_s). We have $G = G_r \cup G_s$ therefore $\Delta(G_r) \leq \Delta(G)$. Using the above lemma we have : $|S_r| \leq 2\Delta(G_r) \leq 2\Delta(G)$. The number of matchings of k edges is at most $\left\lceil \frac{m(G)}{k} \right\rceil \leq \eta_e(G)$ by definition of $\eta_e(G)$. So we have $|S_s| \leq \eta_e(G)$. So in final: $|S| = |S_r \cup S_s| = |S_r| + |S_s| \leq 2\Delta(G) + \eta_e(G) \leq 3\eta_e(G)$. $\square$

So we can conclude that our heuristics will never produce a set of matchings with a number of steps worst as three times the optimal one.

Example:
It is indeed possible to go as near as wanted to the factor 3. This means that 3 is the best factor possible to reach under the given conditions. We are going to illustrate that with some examples.
Take for example $k = 4$, and the graph G of Figure 16. We have here a decomposition in 4 steps. It has been obtained by taking at each step a maximal matching. The optimal decomposition takes only two steps since $\eta_e(G) = \max(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil = \max(2, 2) = 2$. An optimal decomposition is shown in Figure 17.

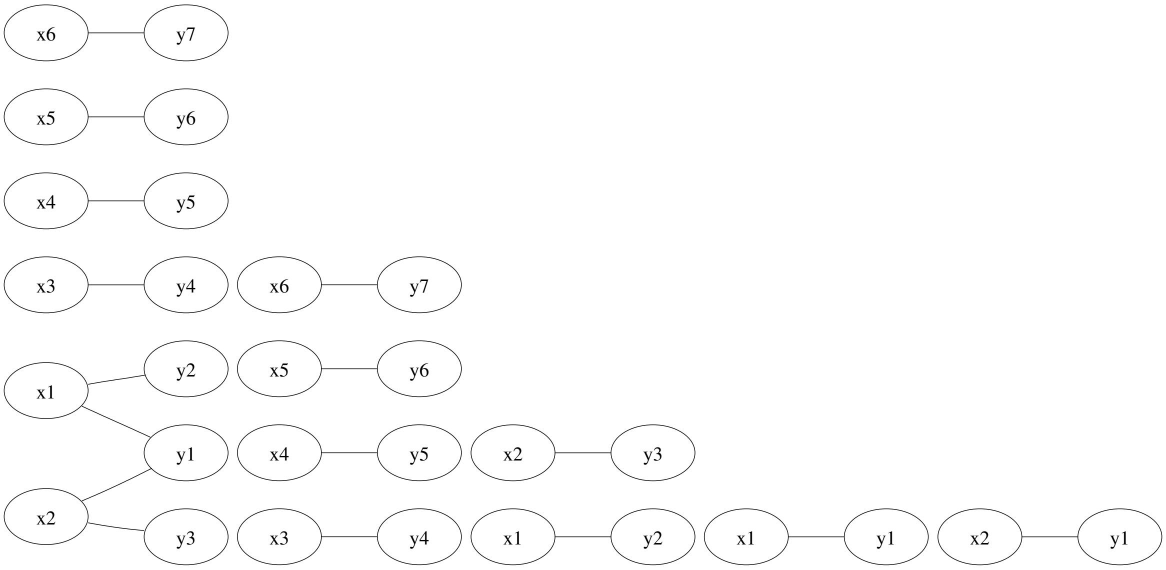

Figure 16. G **followed by the matching forming** G_s **and the three matchings forming** G_r

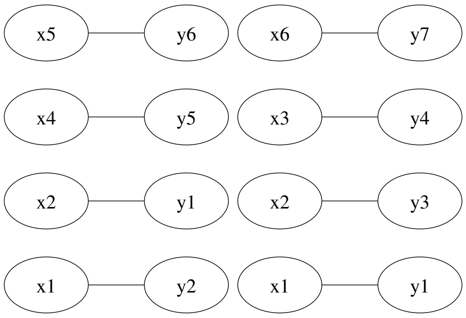

Figure 17. optimal decomposition of G

It is possible to build a family of graphs yielding to bad results. n being an integer, let G_n be defined as follow : we have $|V_1| = (n-1)n^2 + n, |V_2| = (n-1)n^2 + n \times (n-1) + 1$, and the following edges :

1. $(n-1)n^2$ edges which nodes have a degree of 1
2. n edges from $x_1, ..x_n$ (in V_1) to $y1$ (in V_2)
3. for each node $x_1, ..x_n, n-1$ edges to a node in V_2 of degree 1

For G_n we consider $k = n^2$. When $n = 2$ this gives us the bipartite graph from Figure 16. When $n = 3$ this gives the bipartite graph from Figure 18 to which we should add 18 edges incident to nodes of degree 1.

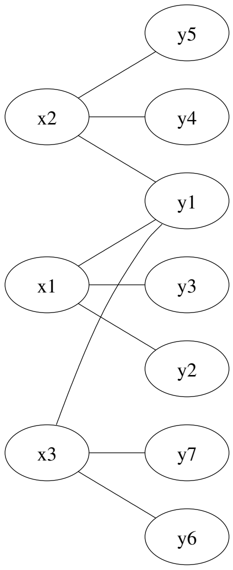

Figure 18. G_3

We now build S_n as follows:

- build $n-1$ matchings of k edges with the edges builded at step 1
- build $n-1$ matchings of $k-1$ edges with the edges builded at step 3
- build n matchings each of 1 edge connecting to y_1 (the edges builded at step 2)

For $n = 2$ this gives the matchings obtained in our example. All matchings taken are maximals at the step they are builded, and we finally have $3n-2$ matchings. $\eta_e(G_n) = \max(\Delta(G_n), \frac{m(G)}{k}) = \max(n, \frac{(n-1)n^2+n+(n-1)n}{n^2}) = \max(n, n) = n$.

Therefore as n increases we may come as close as we want from the factor 3.

6.2. Lower bound on an approximation ratio

We are now going to see a second family of bipartite graphs G'_n, showing it is not possible to prove a bound better than k on the time of the obtained matchings toward the optimal time.

G'_n graphs are defined as follow: $|V_1| = n, |V_2| = n^2$, with the following edges : all nodes in V_1 are of degree n with 1 edge of weight a and all the other edges of weights 1. All nodes in V_2 are of degree 1. For each graph we consider $k = n$.

The first graphs of the family are given in Figure 19.

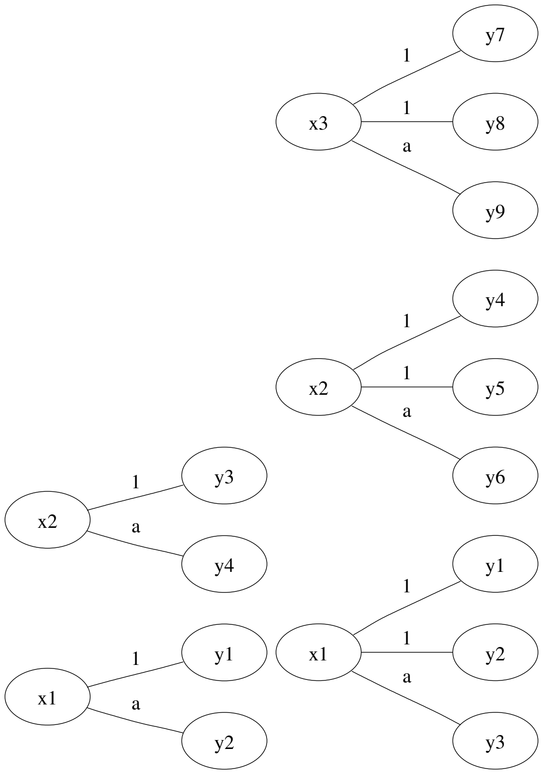

Figure 19. G_2', G_3'

Each matching is taken as follows: take one edge of weight a, and all the other edges of weight 1. Here we reach the minimal number of steps: $\eta_e = n$. But the overall time cost of our matchings is: $a \times n + n\beta$ whereas the optimal time cost is : $\max(W(G'_n), \frac{P(G'_n)}{k}) + n\beta = \max(a + n - 1, a + n - 1) + n\beta = a + n - 1 + n\beta$.

For example, for $n = 2$ the optimal time cost is : $a + 1 + 2\beta$ while the matchings we choose yield to a cost of $2a + 2\beta$ (see Figure 20).

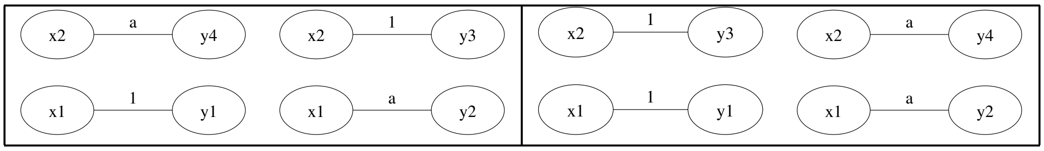

Figure 20. worst case and best case for G'_2

We can see that when a grows, the approximation ratio tends to k. In the general case, where k is unknown, the ratio may be as big as we want. Therefore, the heuristics are not approximation algorithms.

7. New Algorithms

7.1. GGP

7.1.1 Presentation

For our next algorithm, we are going to modify the previous one, to decrease it's complexity. We call this algorithm GGP for Generic Graph Peeling. The main idea is to make the same operations, but to group in one step all identicals matchings. This means we are not going to work on a multigraph. Take for example the Figure 21, with $k = 2$ and $\beta = 1$.

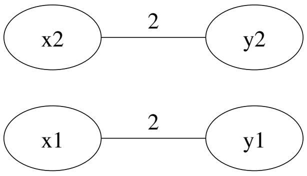

Figure 21. Simple communications

It is obvious that the best communications schedule is to start all communications at the same time and do all of them in only one step i.e. we have only one matching containing all edges as solution of KBPS. But the previous algorithm will give a set of two matchings as solution, as shown in Figure 22.

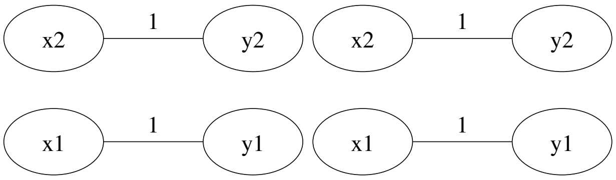

Figure 22. First algorithm's results

In the previous algorithm we were building a regular multigraph, and extracting one matching at each iteration. We are going to do the same, but without building the multigraph. In this algorithm, we build a *weight*-regular bipartite graph (which if turned into a multigraph would be the same as the one builded in first algorithm). On a weight-regular graph, it is always possible to find a perfect matching.So at each main loop iteration, we find a perfect matching. And now to use preemption : we consider the lowest edge's weight in this matching, and reduce all matching's edges by it's value. All edges of weight zero are then removed. As the matching is perfect, the graph remains weight-regular, and we can start again at next iteration.

7.1.2 Algorithm

The algorithm is the following :

1. $R = \emptyset$
2. Build $H = (V_1, V_2, E, f_H)$ such that $\forall e \in E, f_H(e) = \left\lceil \frac{f_G(e)}{\beta} \right\rceil$
3. If $\frac{P(H)}{k} < W(H)$ build the graph $I = (V_{1_I}, V_{2_I}, E_I, f_I)$ such that :
   - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
   - $\forall e \in E, f_I(e) = f_H(e)$
   - $\frac{P(I)}{k} = W(H)$
   - $\exists e \in E_I | e \notin E, f_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall d \in E_I | d \notin E, d \neq e$ we have $f_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall v \in V_{1_I} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_I} | v_2 \notin V_2$

   Else if $\frac{P(H)}{k} \notin \mathbb{N}$
   - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
   - $\forall e \in E, f_I(e) = f_H(e)$
   - $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$
   - $\exists e \in E_I | e \notin E, f_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall d \in E_I | d \notin E, d \neq e$ we have $f_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall v \in V_{1_I} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_J} | v_2 \notin V_2$

   Else $I = H$
4. Build the graph $J = (V_{1_J}, V_{2_J}, E_J, f_J)$ such that :
   - $E_I \subset E_J, V_{1_I} \subset V_{1_J}, V_{2_I} \subset V_{2_J}$
   - $\forall e \in E_I, f_J(e) = f_I(e)$
   - J is $\frac{P(I)}{k}$-weight-regular
   - $\forall (v_1, v_2) \in E_J | (v_1, v_2) \notin E_I,$ either $v_1 \notin V_{1_I}$ or $v_2 \notin V_{2_I}$
5. While $E_J \neq \emptyset$ do :
   (a) Choose a perfect matching M in J
   (b) Add M to S a set of matchings
   (c) Compute s the smallest weight of the edges in M
   (d) For each edge in M reduce it's weight by s
   (e) Remove from E_J all edges of weight 0

6. Remove all edges e in S such that $e \notin E$
7. While $S \neq \emptyset$
   (a) Choose a matching M in S
   (b) Modify f_M such that $\forall e \in M, f_M(e) = \min(f_M(e)\beta, f_G(e))$ and f_G such that $\forall e \in M, f_G(e) = f_G(e) - f_M(e)$
   (c) Add M to R

To build the J graph we use the following algorithm :

- input : $I = V_{1_I}, V_{2_I}, E_I, f_I$ a bipartite graph such that $\frac{P(I)}{k} \in \mathbb{N}$ and $\frac{P(I)}{k} \geq W(I)$
- output : $J = V_{1_J}, V_{2_J}, E_J, f_J$ such that :
  - $E_I \subset E_J, V_{1_I} \subset V_{1_J}, V_{2_I} \subset V_{2_J}$
  - $\forall e \in E_I, f_J(e) = f_I(e)$
  - J is $\frac{P(I)}{k}$-weight-regular
  - $\forall (v_1, v_2) \in E_J | (v_1, v_2) \notin E_I,$ either $v_1 \notin V_{1_I}$ or $v_2 \notin V_{2_I}$
- algorithm :
  1. we first copy the I graph : $V_{1_J} = V_{1_I}, V_{2_J} = V_{2_I}, E_J = E_I$ and $\forall e f_J(e) = f_I(e)$
  2. now we add new nodes in V_{2_J}:
     (a) we call n the node we are currently building. We start with n undefined.
     (b) for each node s in V_{1_J} :
        - while $w(s) \neq \frac{P(I)}{k}$ do :
          * if n is not defined, and a new node n in V_{2_J}
          * add an edge e between n and s with weight $f_J(e) = \min(\frac{P(I)}{k} - w(n), \frac{P(I)}{k} - w(s))$
          * if $w(n) = \frac{P(I)}{k}$ then $n =$ *undef*
  3. now we add new nodes in V_{1_J}:
     (a) we call n the node we are currently building. We start with n undefined.
     (b) for each node s in V_{2_J} :
        - while $w(s) \neq \frac{P(I)}{k}$ do :
          * if n is not defined, and a new node n in V_{1_J}
          * add an edge e between n and s with weight $f_J(e) = \min(\frac{P(I)}{k} - w(n), \frac{P(I)}{k} - w(s))$
          * if $w(n) = \frac{P(I)}{k}$ then $n =$ *undef*
- notes : We can easily see that for each node we will add at most 2 edges.

7.1.3 Properties

Complexity

Let's call U the set of all edges of weight $W(H)$ added at step 3 to build I. We have : $|E_I| \leq |E| + |U| + 1$ (the "1" coming from the edge e). When building J, we need to complete each node's weight to $\frac{P(I)}{k}$. This is done by adding at most two edges to each node. But as all nodes from the edges in U already have the good weight we will not add edges to them and therefore we have :

$$|E_J| \leq |E_I| + 2(|V_{1_I}| - |U|) + 2(|V_{2_I}| - |U|)$$

$$\Rightarrow |E_J| \leq |E_I| + 2(|V_1| + |U| + 1 - |U|) + 2(|V_2| + |U| + 1 - |U|)$$
$$\Rightarrow |E_J| \leq |E_I| + 2(|V_1| + |V_2| + 2)$$
$$\Rightarrow |E_J| \leq |E| + |U| + 1 + 2(|V_1| + |V_2| + 2)$$

When iterating through the main loop at step 5, we remove at least one edge at each step, and *all* edges of U at the last step.

Therefore we have the number of steps $s \leq |E_J| - |U|$

$$\Rightarrow s \leq |E| + 1 + 2(|V_1| + |V_2| + 2)$$

$$\Rightarrow s \leq |E| + 2(|V_1| + |V_2|) + 5$$

So this algorithm achieves polynomial time. Each step costs the time to find a matching which is in :

$$O(\sqrt{n(J)}m(J)) = O(\sqrt{n(G)}(m(G) + n(G)))$$

so we are in :

$$O(\sqrt{n(G)}(m(G) + n(G)) \times (m(G) + n(G))) = O(\sqrt{n(G)}(m(G) + n(G))^2)$$

Approximation factor

For any given input, the output given by GGP may also be obtained by the initial algorithm. We can achieve that by converting a matching of edges of weight w into w matchings with edges of weight 1. Therefore, this algorithm is also a 2-approximation of KBPS.

Reaching worst case

It is possible to reach the approximation factor of 2. Take for example the bipartite graph of Figure 23 with $\beta = 100$ and $k = 2$.

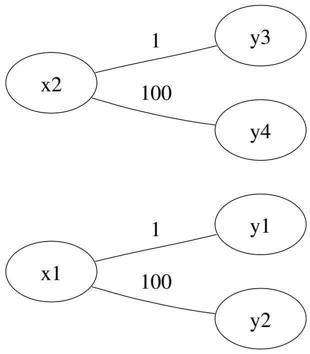

Figure 23. Reaching worst case

Any of the two previous algorithm will compute as shown in Figure 24 and give the result shown in Figure 25.

We can clearly see, that the cost is approximately 4β versus a best case of cost 3β. Indeed it is possible to build a family of cases (see section 6.1) for which the cost will grow until 2.

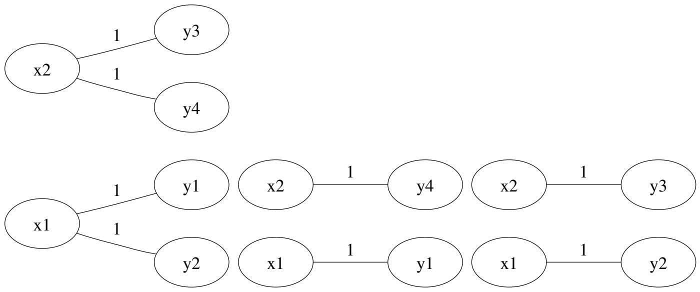

Figure 24. Computing R

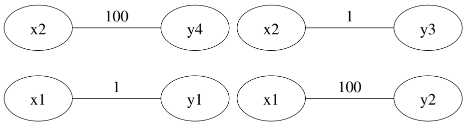

Figure 25. Final scheduling

7.2. OGGP

7.2.1 Presentation

This algorithm is an extension of GGP. We name it OGGP for Optimized Generic Graph Peeling. We are using the fact that in the previous algorithm, at each step, the matching chosen could be any matching. What we want is, at each step, to choose the best possible matching. Intuitively we want to achieve a maximal number of communications in one step. As this number is determined by the *smallest* weight of our matching's edges, we need a way to find the matching with it's smallest edge as large as possible.

7.2.2 Algorithm

First we will describe the algorithm finding a perfect matching with minimum weight maximized. This algorithm is a modification of the Kuhn-Munkres algorithm.

We take as parameter : $G = (V_1, V_2, E, f)$ a weighted bipartite graph, and compute M the perfect matching of G with minimum weight maximized.

1. $G' = \emptyset, M = \emptyset, G'' = G$
2. while M is not perfect in G do:
   (a) choose $e \in E(G'') | \forall e' \in E(G''), f(e) \geq f(e')$
   (b) $E(G') = E(G') \cup e$
   (c) $E(G'') = E(G'') \backslash e$
   (d) M = a maximal matching in G'

Proof by contradiction:

If we call l the last edge added in G', we have $l \in M$ because without l it wasn't possible to find a perfect matching. We also have $\forall e \in G, f(e) > f(l) \Rightarrow e \in G'$. Suppose M' a perfect matching better than M. M' is such that : $\forall e \in M', f(e) > f(l)$. Therefore we have : $M' \subset G'$. This is a contradiction, so M is the perfect matching maximizing the minimum weight.

□

Complexity

This algorithm is in $O(m(G) \times \sqrt{n(G)} m(G)) = O(m(G)^2 \sqrt{n(G)})$.

Main algorithm

We can now write the main algorithm : We take as parameters: G, β, k and return R, a set of matchings.

1. $R = \emptyset$
2. Build $H = (V_1, V_2, E, f_H)$ such that $\forall e \in E, f_H(e) = \left\lceil \frac{f_G(e)}{\beta} \right\rceil$
3. If $\frac{P(H)}{k} < W(H)$ build the graph $I = (V_{1_I}, V_{2_I}, E_I, f_I)$ such that :
   - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
   - $\forall e \in E, f_I(e) = f_H(e)$
   - $\frac{P(I)}{k} = W(H)$
   - $\exists e \in E_I | e \notin E, f_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall d \in E_I | d \notin E, d \neq e$ we have $f_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall v \in V_{1_I} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_I} | v_2 \notin V_2$

   Else if $\frac{P(H)}{k} \notin \mathbb{N}$
   - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
   - $\forall e \in E, f_I(e) = f_H(e)$
   - $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$
   - $\exists e \in E_I | e \notin E, f_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall d \in E_I | d \notin E, d \neq e$ we have $f_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
   - $\forall v \in V_{1_I} | v \notin V_1$ the degree of v is 1, the edge e incident to v is incident to $v_2 \in V_{2_J} | v_2 \notin V_2$

   Else $I = H$
4. Build the graph $J = (V_{1_J}, V_{2_J}, E_J, f_J)$ such that :
   - $E_I \subset E_J, V_{1_I} \subset V_{1_J}, V_{2_I} \subset V_{2_J}$
   - $\forall e \in E_I, f_J(e) = f_I(e)$
   - J is $W(I)$-weight-regular
   - $\forall (v_1, v_2) \in E_J | (v_1, v_2) \notin E_I,$ either $v_1 \notin V_{1_I}$ or $v_2 \notin V_{2_I}$

   We use for that the same algorithm as in GGP.
5. Build $f_q : E_J \to Q | \forall e \in E, f_q(e) = \frac{f_G(e)}{\beta}, \forall e \in E_J | e \notin E, f_q(e) = f_J(e)$
6. While $E_J \neq \emptyset$ do :
   (a) Choose a perfect matching M in J using the preceding algorithm, and f_q as weight function (f_q is used only for the selection)

(b) Add M to S a set of matchings
(c) Compute s the smallest weight of the edges in M
(d) For each edge in M reduce it's weight by s
(e) For each edge e in M, $f_q(e) = \min(f_q(e) - s, 0)$
(f) Remove from E_J all edges of weight 0

7. Remove all edges e in S such that $e \notin E$
8. While $S \neq \emptyset$
   (a) Choose a matching M in S
   (b) Modify f_M such that $\forall e \in M, f_M(e) = \min(f_M(e)\beta, f_G(e))$ and f_G such that $\forall e \in M, f_G(e) = f_G(e) - f_M(e)$
   (c) Add M to R

7.2.3 Properties

Complexity

The matching algorithm has a complexity of $O(m(J)^2\sqrt{n(J)}) = O((m(G)+n(G))^2\sqrt{n(G)})$, so we have as complexity: $O((m(G)+n(G))^2\sqrt{n(G)} \times (m(G)+n(G))) = O((m(G)+n(G))^3\sqrt{n(G)})$

Approximation factor

As a direct extension of GGP, OGGP is also a 2-approximation of KBPS. We have not found a family of examples leading to the worst ratio of 2.

In particular, we can see that on the examples of section 7.1.3, it would not be possible to reach the worst case as the largests communications would be grouped together.

7.3. Simulations

The experiments presented in this section are differents tests of our algorithms on random graphs. All tests have been conducted on graphs only and no real redistribution is taking place.

The algorithms have been implemented in a library using C++ and the libstdc++. The bipartites graphs used as input are totally random, but in our tests, we will consider $|V_1| = |V_2| = 20, 150 \leq |E| \leq 300$. Small tests with differents parameters have led to the same kind of results. Each point in the results graphs is computed using the average of 100000 tests. As GGP is giving the same results as the first algorithm, we will not conduct any experiments on the first algorithm.

7.3.1 Comparing heuristics with GGP

Tests on small weights

The graphs of Figure 26 displays how the approximation ratio varies when k grows. The weights are generated randomly between 1 and 20 and $\beta = 1$.

We can see that the algorithm is giving better results than the heuristics, and that they can give an approximation ratio worst than 2.

Tests on big weights

The graphs of Figure 27 displays how the approximation ratio varies when k grows. The weights are generated randomly between 1 and 100000 and $\beta = 1$.

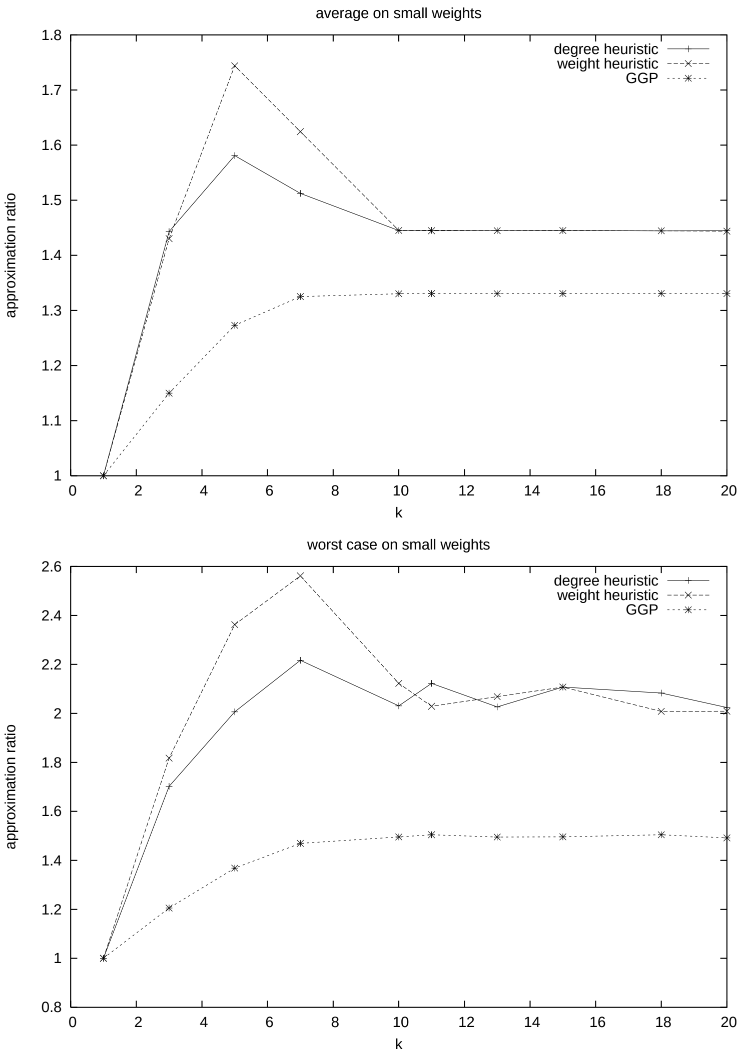

Figure 26. results on small weights

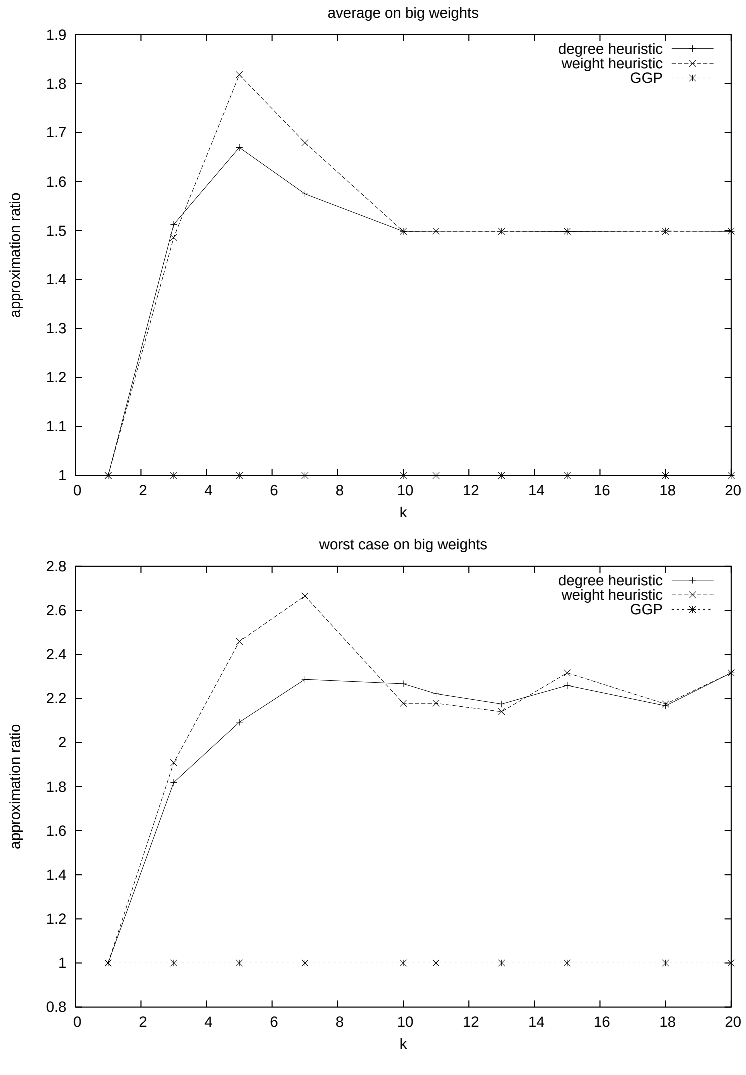

Figure 27. results on big weights

The huge difference between the algorithm and the heuristics is essentially given by the use of preemption. As the cost of splitting a communication (β) is weak when compared to the cost of communications, we can achieve far better results.

7.3.2 Comparing GGP and OGGP

Tests on small weights

The graph of Figure 28 displays how the approximation ratio varies when k grows. The weights are generated randomly between 1 and 20 and $\beta = 1$.

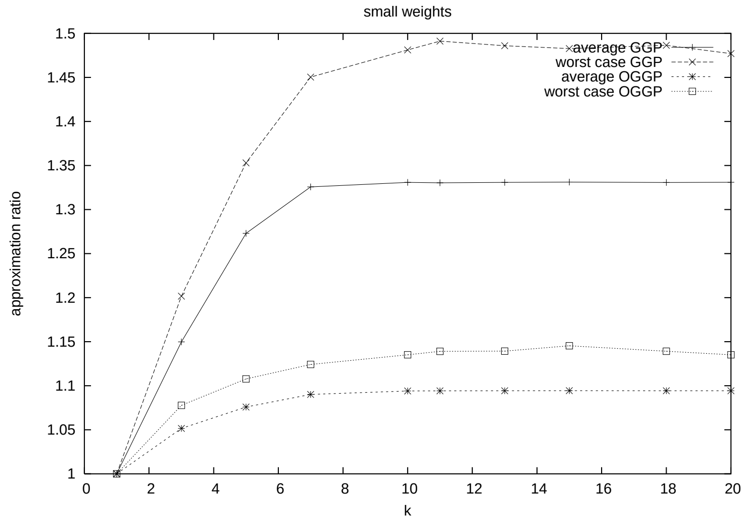

Figure 28. small weights

We can see that as k grows the approximation ratio grows and stabilizes. Also we can see that the approximation ratio never reaches 2, and that OGGP always gives better results than GGP.

Tests on big weights

The graph of Figure 29 displays how the approximation ratio varies when k grows. The weights are generated randomly between 1 and 100000, $\beta = 1$.

We can see that we have the same type of graph as on small weights, but with an approximation ratio far closer from 1. This reduces the difference between the two algorithms.

Tests on beta

The graph of Figure 30 displays how the approximation ratio varies when $beta$ grows. The weights are generated randomly between 1 and 20, and for each point of the graph, k varies between 1 and 20. This test configuration has been the one giving the worst results on our "hand-made" tests.

We can see that OGGP is still giving better result than GGP, β is affecting results until it becomes bigger than all weights. And the results are worse than in other cases, with OGGP giving on certain exemples an approximation ratio of 1.5.

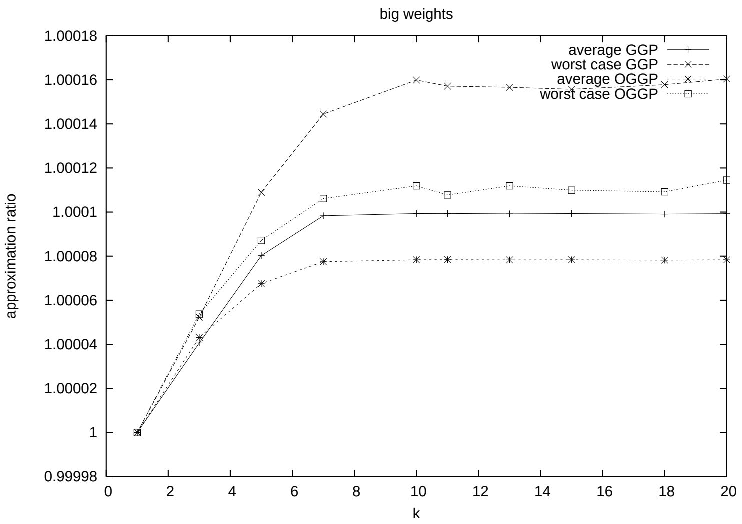

Figure 29. big weights

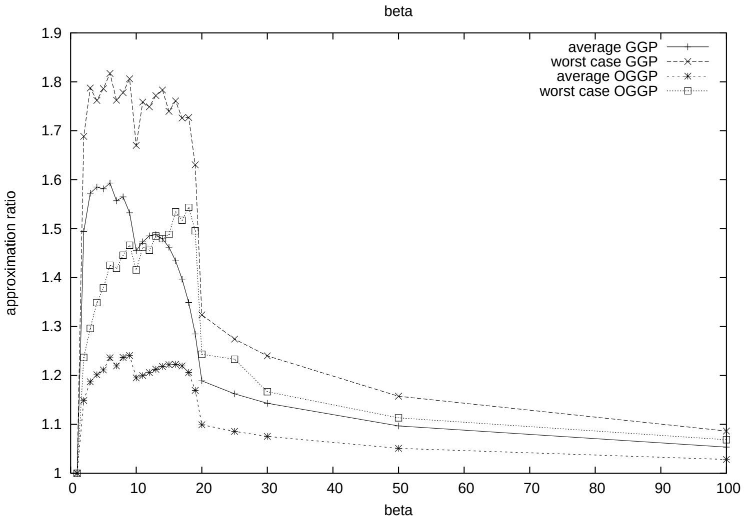

Figure 30. beta

7.4. Experimental results

We have completed a first set of experiments on real clusters. Our configuration is the following: two clusters of 10 machines each, connected by two 100Mbits switchs, and a 100Mbits link. All network cards are 100Mbits cards. All machines are on the local network, so latency is in the order of milliseconds. To focus on the interesting cases where $k \neq 1$ we are using *rshaper* [17] under linux to limit artificially each's card bandwidth (both upload and download) to 20Mbits, 33Mbits, and 15Mbits. With these configurations we have : $k = 5$, $k = 3$, $k = 7$ and $beta = 1$. We should note the results should be differents on a bigger network than a LAN.

All algorithms have been implemented using MPI. Each communication is done synchronously and communication steps are synchronized using barriers. Algorithms are compared to a raw approach where all communications are issued simultaneously (and therefore asynchronously) in only one step of communication.

We take as input an all to all communication (i.e. a complete bipartite graph) with random data size uniformly distributed between 10 MB and n MB. We plot the total communication time obtained when n increases from 10 to 80. The results for $k = 3$, $k = 5$, $k = 7$ are displayed respectively in Figures 31, 32, 33.

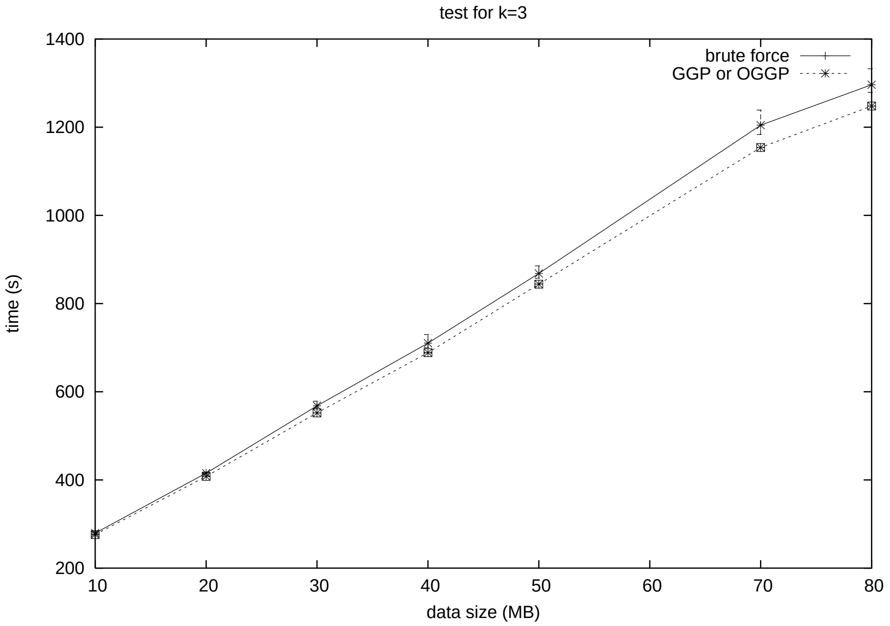

Figure 31. $k = 3$

Several observations can be made:

- We achieve a 5% to 20% reduction of communications costs. Although we are alone on a local network, where TCP is efficient, we are able to achieve better results.
- The barriers cost extremely little time. Although the OGGP gives 50% less steps of communication as GGP, they generate the same communication time. However we believe the cost of synchronisations may increase if we introduce some random perturbations on the network.
- The brute-force approach does not behave deterministicaly. When conducting several time the same experiments we see a time variation of up to 10 percents. It is interesting to see that our approach on the opposite behaves deterministically.
- As the available bandwidth decreases (i.e. k increases) we increase the benefits of using GGP or $OGGP$ over the brute-force approach.

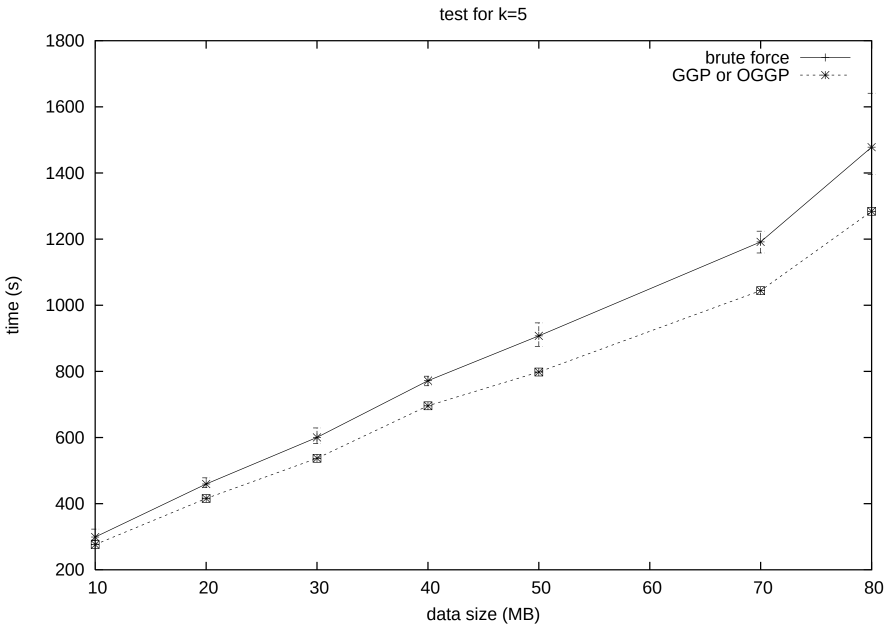

Figure 32. $k = 5$

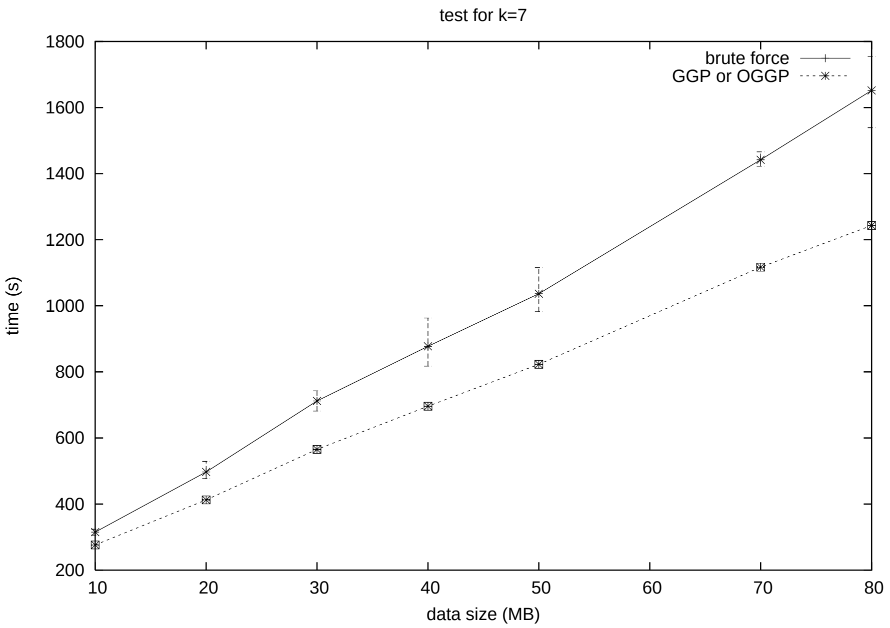

Figure 33. $k = 7$

8. Conclusion

In order to optimize redistribution time, we have mapped network constraints and communications to a mathematical model using bipartite graphs. We introduced KBPS as the NP-complete problem corresponding to the real-life problem we try to solve. We studied two heuristics and one approximation algorithm and demonstrated their advantages and limits. We then introduced two new 2-approximations algorithms called GGP and OGGP, with polynomial complexity. To evaluate the algorithms we first executed some experiments on bipartite graphs. They showed overall good results, with an approximation attaining at worst 1.5 for the third algorithm. Finally we conducted some real world experiments on a local area network. These experiments showed the possibility to obtain better results using message scheduling than relying on a raw approach.

Future work will include some tests on high speed networks, where TCP's work is less efficient. We will also conduct experiments with random network parameters variation, to see how the differents algorithms are able to handle the load. On a more theoretical aspect, we will study the possibility to issue first a local redistribution inside a cluster before starting to send data accross the network. This should enable to take advantage of local connections of higher speed than the distant ones.

Bibliography

References

[1] F. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. In *AlgoTel 2002*, Mèze, France, May 2002.

[2] P. B. Bhat, V. .K. Prasanna, and C. .S. Raghavendra. Block Cyclic Redistribution over Heterogeneous Networks. In *11th International Conference on Parallel and Distrinuted Computing Systems (PDCS 1998)*, 1998.

[3] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders. *IEEE Transactions on Communications*, 29(5):721–726, 1981.

[4] H. Choi, H.-A. Choi, and M. Azizoglu. Efficient Scheduling of Transmissions in Optical Broadcast Networks. *IEEE/ACM Transaction on Networking*, 4(6):913–920, December 1996.

[5] Johanne Cohen, Emmanuel Jeannot, and Nicolas Padoy. Messages Scheduling for Data Redistribution between Clusters. In *Algorithms, models and tools for parallel computing on heterogeneous networks (HeteroPar'03) workshop of SIAM PPAM 2003*, Czestochowa, Poland, September 2003. To appear in LNCS series.

[6] P. Crescenzi, D. Xiaotie, and C. H. Papadimitriou. On Approximating a Scheduling Problem. *Journal of Combinatorial Optimization*, 5:287–297, 2001.

[7] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. *IEEE Transaction on Parallel and Distributed Systems*, 9(2):192–205, 1998.

[8] A. Ganz and Y. Gao. A Time-Wavelength Assignment Algorithm for WDM Star Network. In *IEEE INFOCOM'92*, pages 2144–2150, 1992.

[9] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos. CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11(3):224 – 236, August 1997.

[10] I. S. Gopal, G. Bongiovanni, M. A. Bonuccelli, D. T. Tang, and C. K. Wong. An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams. *IEEE Transactions on Communications*, COM-30(11):2475–2481, November 1982.

[11] I.S. Gopal and C.K. Wong. Minimizing the number of switching in an ss/tdma system. *IEEE Trans. on Communications*, 1885.

[12] Oak Ridge National Labs. Mxn. `http://www.csm.ornl.gov/cca/mxn`.

[13] M. Mishra and K. Sivalingam. Scheduling in WDM Networks with Tunable Transmitter and Tunable Receiver Architecture. In *NetWorld+Interop Engineers Conference*, Las Vegas, NJ, May 1999.

[14] N. Padoy. Redistribution de données entre deux grappes d'ordinateurs. Rapport de stage, de l'École Normale Supérieure de Lyon, 2002.

[15] G. R. Pieris and Sasaki G.H. Scheduling Transmission in WDM Broadcast-and-Select Networks. *IEEE/ACM Transaction on Networking*, 2(2), April 1994.

[16] N Rouskas and V Sivaraman. On the Design of Optimal TDM Schedules for Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies. In *IEEE INFOCOM'96*, pages 1217–1224, 1996.

[17] A. Rubini. Linux module for network shaping
`http://ar.linux.it/software/#rshaper`.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399