An Alternative Implementation for the FPT k-Vertex Cover Parallel Algorithm*

Henrique Mongelli, Deiviston S. Aguena,
Edson N. Cáceres
Universidade Federal de Mato Grosso do Sul
Campo Grande-MS, Brazil
{mongelli,edson}@dct.ufms.br,
deiviston@gmail.com

Siang Wun Song
Universidade de São Paulo
Instituto de Matemática e Estatística
São Paulo - SP, Brazil
song@ime.usp.br

Abstract

Adequate choice of data structures and special effort in implementation are crucial to the good performance of parallel algorithms. In this paper, we present experimental results of a BSP/CGM implementation for the FPT (Fixed-Parameter Tractability) Vertex Cover problem, also known as k-Vertex Cover. We propose an alternative implementation that has as its basis an algorithm that combines the parallel FPT algorithm proposed by Cheetham et al. and the Downey's et al. sequential FPT algorithm. Previously, a better and refined implementation, based on the Cheetham et al. Algorithm was presented by Hanashiro. In his experiments, Hanashiro obtained better results than those presented by Cheetham et al. In this paper, implemented the new adapted algorithm for the k-Vertex Cover and compared our experimental results with those of Hanashiro et al, using the same input data (conflict graphs of amino acids). We report substantial improvement over the results of Hanahiro et al, with speedups from 3 to 20 times relative to that implementation.

1. Introduction

Choice of adequate data structures is one of the important issues in the design of parallel algorithms. The space required to store the necessary data, as well as the access time of the data structure are two important measures. The space requirement is particularly important if data structures are to be transmitted among processors, since it would then affect also the overall execution time of the parallel algorithm. On the other hand, it is not sufficient to have a space efficient data structure that does not provide quick access time. Often there is no single data structure that minimizes both measures and a solution of compromise is often necessary. A good example to illustrate this is the proposal of a data structure that was able to give considerable improvement on the performance of an algorithm to solve the the longest subsequence problem of two given strings, as well as some other related problems (e.g. the string editing problem) [2].

In this paper we discuss the choice of data structure and implementation issues to obtain an efficient execution of a coarse-grained parallel algorithm to solve the k-Vertex Cover problem. This problem has important applications, for example, in the analysis of alignment of multiple sequences in Computational Biology [6, 18]. Moreover, this was one of the first problems proved to be fixed-parameter tractable (FPT), a technique to deal with some of the so-called intractable problems. Given a graph $G = (V, E)$, where V is a set of vertices and E a set of edges, and an integer k, the k-Vertex Cover problem determines if there exists a subset of vertices V' of V of maximum size k, such that every edge of E has at least one vertex in V'.

Dealing with intractability has been one of the most important problems in theory and practice in Computer Science. With the objective of attempting to deal with intractability, methods that use approximation, randomization and heuristics have been considered [19]. However, such methods present the disadvantages of not always offering guarantees about performance or exactness in the solution. The Parameterized Complexity, developed by Downey and Fellows [9, 10, 11, 13], is an alternative to those methods. More recent works explore the use of parameterization, combined with heuristic and parallelism [5, 14, 16].

Consider a problem whose input size is n. Problems are said to be parameterizable if they can be divided in two parts: the main part (n) and one parameter (k) [13]. A parameterizable problem is fixed-parameter tractable (FPT) if it can be solved by an algorithm in time *O(f(k)n^{α})*, where f is an arbitrary function and α is a constant independent of

*Partially supported by FAPESP Proc. No. 2004/08928-3 CNPq Proc. Nos. 55.0895/07-8, 30.5362/06-2, 30.2942/04-1, 62.0123/04-4, 48.5460/06-8, 62.0171/06-5 and FUNDECT 41/100.115/2006.

n and k [13]. The main idea is the following. Consider an intractable problem of input size n that can be solved by an exponential time algorithm (in n). If the problem is parameteriable, there is a fixed parameter k and it can be solved in time *O(f(k)n^{α})*, which may be polynomial in n and exponential in k. That is, the exponential nature of the solution lies in the parameter k, and not in n. In practice, depending on the problem, parameter k may be *small* compared to n and the exponential nature of the solution may be tractable. This is the case in several FPT algorithms, as is the case of the the k-Vertex Cover problem. Notice that a FPT algorithm does not merely gives an approximate solution, but rather it solves the problem exactly.

FPT algorithms for the k-Vertex Cover problem have been proposed by Cheetham *et al.* [6] and Downey *et al* [12]. Hanashiro [16] presented a refined and improved implementation of the Parallel FPT Algorithm proposed by Cheetham *et al.* [6]. The results presented by Hanashiro have surpassed the results obtained by Cheetham *et al.*, even using an inferior computational environment. In this paper, we propose a even more efficient implementation for the k-Vertex Cover problem that has as its basis an algorithm that combines the ideas presented by Cheetham *et al.* [6] and by Downey *et al* [12]. In our experiments, we use as input data the same conflict graphs of amino acids used by Hanashiro and Cheetham *et al*. For three of these graphs, we obtained better results, in relation to the Hanashiro implementation, using the same computational environment. The contribution in this work is in the proposal of an adapted Parallel FPT Algorithm and its efficient implementation through an appropriate choice of data structures. This algorithm forms the basis for other alternative parallel FPT algorithms and their implementations for the k-Vertex Cover problem.

This work is organized as follows. In Section 2 we present the Coarse-Grained Multicomputer model used in this work. In Section 3 we present concepts of Parameterized Complexity and FPT, with some of more technical detailis presented in the Appendix. In Section 4, we present the k-vertex Cover problem. In Section 5 we give the FPT algorithms to solve this problem. We also present an improved algorithm of our implementation that combines the algorithms of Cheetham *et al.* and Downey *et al*. This algorithm will be referred to it as the Adapted Cheetham *et al.* Algorithm. In Section 6, we present experimental results and, finally, in Section 7 we conclude and discuss future works.

2 Coarse-Grained Multicomputer Model

The *Coarsed-Grained Multicomputer* (CGM) model was proposed by Dehne *et al* [8] and consists of a set of p processors, each one with local memory of size $O(n/p)$ and connected by some interconnection network, where n is the size of the problem and $n/p \geq p$. This model, to be referred to as BSP/CGM, is a simplification of the *Bulk Synchronous Parallel* (BSP) model proposed by Valiant [21], one of the first realistic models proposed in the literature. The realistic models define parameters to map the main characteristics of parallel machines, and take into consideration, among other things, the communication time among processors.

In a BSP/CGM algorithm local computation rounds alternate with global communication among processors. In a computation round, each processor executes a sequential algorithm on its local data. In a communication round, each processor can send and receive $O(n/p)$ data. Synchronization barriers separate local computation rounds from communication rounds.

The sum of times expended with local computation as well as with communication among processors defines the execution time of the BSP/CGM algorithm. The goal is to design a BSP/CGM algorithm that minimizes both the number of communication rounds and the local computation time.

3. Parameterized Complexity and FPT

As mentioned earlier, the main idea of fixed-parameter tractability (FPT) is that if an intractable problem is parameterizable, then there is a fixed parameter k such that the problem can be solved in time *O(f(k)n^{α})*. That is, the exponential nature of the solution may be in the parameter k, and not in n. In practice, depending on the problem, parameter k may be *small* compared to n and the exponential nature of the solution may be tractable. This is the case in several FPT algorithms, as is the case of the the k-Vertex Cover problem.

Two techniques are usually applied in the FPT algorithms design: the reduction to problem kernel and the bounded search tree. These techniques can be combined to solve the problem.

The appendix contains more technical details concerning FPT.

4 Vertex Cover problem

The Vertex Cover problem is a central problem in Computer Science based on the following considerations [18].

- It was among the first NP-complete problems.
- There have been numerous efforts to design approximation algorithms, but it is also known to be hard to approximate.
- It is of central importance in parameterized complexity theory and has one of the most efficient FPT algorithms.

- It has important applications, e.g., in Computational Biochemistry, where it is used to solve conflicts between sequences.

A vertex cover for a graph $G = (V, E)$ is a set $V' \subseteq V$ of vertices such that, for each edge $(u, v) \in E$, at least one of the vertices u or v belongs to V'. Figure 1 presents a vertex cover $V' = \{v_2, v_5, v_6, v_7\}$ of size $k = 4$.

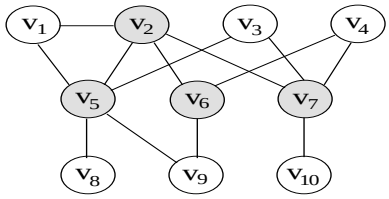

Figure 1. The cover $V' = \{v_2, v_5, v_6, v_7\}$ is one of possible vertex covers.

Definition 1 *k-Vertex Cover Problem*
Instance: *Graph $G = (V, E)$ and a positive integer k.*
Parameter: *k.*
Question: *Does the Graph G have a vertex cover with size less or equal to k?*

5 FPT Algorithms for the k-Vertex Cover Problem

Several sequential FPT algorithms for the k-Vertex Cover Problem have been proposed. See Table 1.

Author(s)	Time Complexity
Buss [4]	$O(kn + (2k^2)^k k^2)$
Downey e Fellows [9]	$O(kn + 2^k k^2)$
Balasubramaniam $etal.$ [3] (B1)	$O(kn + \sqrt{3^k} k^2)$
Balasubramaniam $etal.$ [3] (B2)	$O(kn + 1.324718^k k^2)$
Downey, Fellows e Stege [12]	$O(kn + 1.31951^k k^2)$
Niedermeider e Rossmanith [18]	$O(kn + 1.29175^k k^2)$
Fellows e Stege [20]	$O(kn + max(1.25542^k k^2, 1.2906^k 2.5k))$
Chen, Jia e Kanj [7]	$O(kn + 1.271^k k^2)$

Table 1. FPT sequential algorithms with their respective times

The first parallel FPT algorithm was published by Cheetham, Dehne, Rau-Chaplin, Stege and Taillon [6], to be referred to as the Cheetham *et al* Algorithm. It requires $O(\log p)$ communication rounds. The algorithm has two phases: in the first phase a reduction to problem kernel is applied; the second phase consists of building a bounded search tree that is distributed among the processors.

In this paper we propose an improved algorithm, called the Cheetham *et al.* Adapted algorithm. It is based on several other algorithms which we now describe in the next subsections. The proposed Cheetham *et al.* Adapted algorithm is presented in subsection 5.6.

5.1. Buss Algorithm

The Buss Algorithm [4] receives a graph $G = (V, E)$ and an integer k. This algorithm describes a method of reduction to the problem kernel that reduces in polynomial time the graph G in G', being that the size of G' is limited by a function of the parameter k.

The Buss Algorithm is based on the idea that all the vertices of degree greater than k belong to any vertex cover for graph G of size smaller or equal to k. Therefore, such vertices must be added to the partial cover and removed from the graph. If there are more than k vertices in this situation, there is no vertex cover of size smaller or equal to k for the graph G.

In case there are more than k vertices of degree greater or equal to k, it will not have a vertex cover for G of maximum size k. From this moment on, if the number of edges is less then $k.k'$, we apply a brute force algorithm to determine if there is a vertex cover for G' of size less or equal to k', for the instance (G', k') and vertex partial cover H.

5.2. The Balasubramanian *et al.* B1 Algorithm

Balasubramanian *et al.* presented two algorithms to find the k-Vertex Cover of a given a graph $G = (V, E)$ and an integer k. The first Balasubramanian *et al.* algorithm [3], that we call *B1 Algorithm*, applies first the reduction to the problem kernel method, generating the instance (G', k') and a partial covering H.

It then generates a ternary limited search tree with the root node containing the instance (G', k') and H, generated in the previous step. Notice that we only generate the nodes along the depth-first search path.

Each node of the tree stores a vertex partial cover H and a reduced instance of the problem $(G'' = (V'', E''), k'')$. The graph G'' results from the removal of the incident edges in H and any isolated vertex, and the number k'' is the maximum size of the vertex cover G''.

The search tree has the following property: for each existing vertex cover for graph G of size smaller or equal to k, there exists a corresponding tree node with an empty graph and a vertex cover (not necessarily the same) of size smaller or equal to k. However, if there is no vertex cover of size smaller or equal to k for graph G, then no tree node possesses a resulting empty graph.

During the depth-first search we choose a vertex $v \in V''$ that passes through at most three edges, with the following possible paths: simple paths of length three, cycles of length three, path of length two, and paths of length one. Each one of the two first paths originates three cover possibilities and the last two reduce the graph inside its own node.

The tree interrupts the growth when the node that is being processed has an instance with a partial cover of size less or equal to k and with an empty resultant graph, or when we cover the entire tree, guaranteeing the limit of k.

5.3. The B2 Balasubramanian *et al.* Algorithm

The second Balasubramanian et al. algorithm [3], that we call *B2 Algorithm*, applies the reduction to the problem kernel method and generates the instance (G', k') and a partial cover H. It then generates a limited search tree T, which is not necessarily ternary. Tree T will have its root node containing the instance (G', k') and H, generated in the previous step.

In the depth-first search we choose a vertex v from the graph G'' and we have the following paths, in order of priority: paths with vertex of degree one, paths with vertex of degree two, paths with vertex of degree greater or equal to five, paths with vertex of degree three and paths with vertex of degree four. According to the degree and the choice of the vertex we have several possibilities for the cover, each one of them from the source to a branch of the limited search tree.

5.4 The Cheetham *et al.* Algorithm

The algorithm of Cheetham *et al.* [6] parallelizes both phases: the reduction to problem kernel and bounded search tree. In the first phase, we use the Buss algorithm [4]. In the parallel version, each processor $p_i, 0 \leq i \leq p-1$ computes the degree of the vertices labeled as $i * (n/p)$ to $((i+1)*(n/p))-1$ and receives m/p edges from the list of edges of the graph G. The edges are then sorted by the first vertex. Each processor informs the other processors which local vertices have degree greater than k. In this way all processors will be capable of removing these edges. Then each processor will exchange messages, informing the edges obtained.

In the phase of the bounded search tree, using B1 Algorithm [3], we generate the complete ternary tree T with height $h = \log_3 p$ and p leaves (denoted y_o to y_{p-1}) (see Figure 2), where the root node stores the partial vertex cover and the instance (G', k'). Observe that at the begin of this phase, all the processors have the same instance obtained at the previous phase. The tree T is not created explicitly by the processors. Each processor $P_i, 0 \leq i \leq p-1$ considers the unique path between the root and the leaf y_i, where $i/(p/3^{h+1})$. The B1 Algorithm is interrupted when some node reaches the level $\log_3 p$ (leaf y_i).

With this, we finish the first part of this phase. In the second part, each processor P_i executes locally the instance (G''_i, k''_i) corresponding to the leaf y_i of the tree T using the B2 Algorithm, as presented in Figure 2. This phase will finish either if we find a vertex cover for G of size less or equal k, or if all the processors have traversed the entire tree, meaning that it was not possible to find a valid cover.

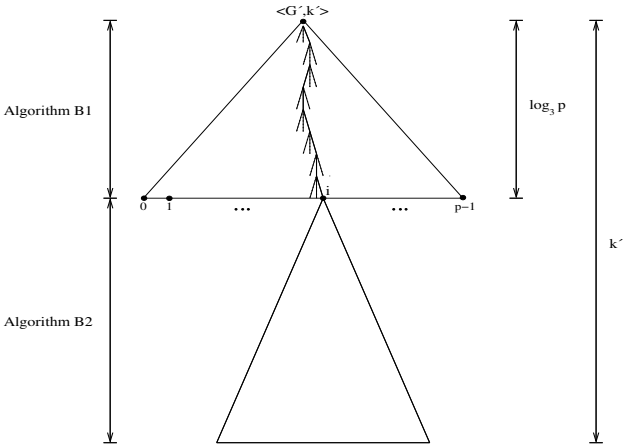

Figure 2. Processor P_i traversed the path from the root to leaf y_i using B1 Algorithm. After that, P_i calculates the entire subtree whose root is y_i using B2 Algorithm.

The Cheetham *et al.* algorithm can be adapted by using other FPT sequential algorithms, in each one of the parts of the bounded search tree phases. Notice that the first part restricts the use of algorithms that generate strictly ternary trees. The second part is more flexible and constitutes the basis of our adapted algorithm.

5.5 The Downey *et al.* Algorithm

In this section, we present the Downey *et al* algorithm [13] for the k-Vertex Covert problem, to be referred to from now on as the Dk-VC algorithm. It is based on both techniques *reduction to problem kernel* and *bounded search tree*. This algorithm consist of two phases. The first phase computes the *kernel* of an given instance (G, k), or answer "no" if it does not exist.

The kernel is an instance (G', k') where $|G'| < k^2$ and $k' \leq k$, such that G' has a vertex of size k' if and only if G has a cover of size k. The reduction of (G, k) to (G', k') can be computed in $O(kn)$ time, where k is the number of vertices in G. In the second phase we build the search tree of maximum height k and the root is labeled as (G', k').

Phase 1 (Reduction to problem kernel): Starting with (G, k) and, while possible, apply the following reduction rules:

1. if G has a vertex v with degree greater than k, then replace (G, k) by $(G - v, k - 1)$.
2. If G has two nonadjacent vertices u, v, such that $|N(u) \cup N(v)| > k$, then replace (G, k) by $(G+uv, k)$.
3. If G has adjacent vertices u, v such that $N(v) \subseteq N[v]$, then replace (G, k) by $(G - u, k - 1)$.
4. If G has a pendant edge uv with u having degree 1, then replace (G, k) by $(G - uv, k - 1)$.
5. If G has a vertex x of degree 2, with neighbors a and b, and none of the above cases applies, then replace (G, k) by (G', k), where G' is obtained from G, deleting the vertex x, and adding the edge ab, and adding all possible edges between a, b and $N(a) \cup N(b)$.
6. If G has a vertex x of degree 3, with neighbors a, b, c, and none of the above cases applies, then replace (G, k) by (G', k), where G' is obtained from G according to the following cases depending on the number of edges between a, b and c:

   6.1 There are no edges between a, b e c. In this case, G' is obtained from G by deleting the vertex x, adding edges from c to all vertices in $N(a)$, adding edges from a to all vertices in $N(b)$, adding edges from b to all vertices in $N(c)$ and adding edges ab and bc.

   6.2 There is exactly one edge in G between vertices a, b, c, assumed to be the edge ab. In this case G' is obtained from G by deleting x from G, adding edges from c to all vertices in $N(b) \cup N(a)$, from a to all vertices in $N(c)$, from b to all vertices in $N(c)$, and adding edges bc and ac.

At the end of *Phase 1* we have reduced (G, k) to (G', k') and G' has minimum degree 4, if we have not already answered the question. The answer is "no" if G' is more than k^2.

Phase 2 (Search Tree):

In this phase we build a search tree of height k where the root is labeled with the output of phase 1 (G', k'). The branch procedure is performed if there are vertices of degree at least 6. The reductions of Phase 1 are applied in each branch. We can assume that each leaf of the resulting search tree is a graph with degree 4 or 5.

- **Degree 4 vertices**: If there is a vertex x of degree 4, then suppose that the neighbors are a, b, c, d. We consider various cases according to the number of edges present between the vertex a, b, c e d. Note that if not all a, b, c, d are in a vertex cover C, then we can assume that at most two of them are.

  Case 1: The subgraph induced by the vertices a, b, c, d has an edge, say ab. Then c e d together cannot be in a cover unless all four a, b, c and d are there. We branch accordingly: (i) $a, b, c, d \subseteq C$, (ii) $N(c) \subseteq C$, (iii) $c \cup N(d) \subseteq C$..

  Case 2: The subgraph induced by a, b, c, d is empty. We consider three subcases:

  Subcase 2.1 Three of vertices (say a, b, c) have a common neighbor y other than x. We branch accordingly: (i) $a, b, c, d \subseteq C$, (ii) $x, y \subseteq C$.

  Subcase 2.2 If Subcase 2.1 does not hold, there may be a pair of vertices that have a total of six neighbors other than x. Let this pair be a and b. If all of a, b, c, d are not in the vertex cover C, or $c \notin C$, or $c \in C$ and $d \notin C$, or both $c \in C$ and $d \in C$ (in which case $a \notin C$ and $b \notin C$). We branch accordingly: (i) $a, b, c, d \subseteq C$, (ii) $N(c) \subseteq C$, (iii) $c \cup N(d) \subseteq C$, (iv) $c, d \cup N(a, b) \subseteq C$.

  Subcase 2.3 If Subcases 2.1 and 2.2 do not hold, then the graph must have the following structure in the vicinity of x: (1) x has four neighbors a, b, c, d and each these has degree 4. (2) There is a set E of six vertices, such that each vertex in E is adjacent to exactly two vertices in a, b, c, d, and the subgraph induced by $E \cup a, b, c, d$ is K_4. In this case we branch according to: (i)$a, b, c, d \subseteq C$, (ii) $(E \cup x) \subseteq C$.

- **Degree 5 vertices**: If graph is regular of degree 5 and none of the reduction rules of Phase 1 can be applied, then we choose a vertex x of degree 5 and do the following. First, we can branch from (G, k) to $(G-x, k-1)$ and $(G-N[x], k-5)$. Then we choose a vertex u of degree 4 in $G-x$ and branch according to one of above cases. The result of these two combined steps is that from (G, k) we created a subtree where one of following cases hold: (i) there are four children with parameter $k - 5$, from Case 1, (ii) there are three children with parameter $k_1 = k - 5, k_2 = k - 5$ and $k_3 = k - 3$, from Subcase 2.1, (iii) there are five children with parameter $k_1 = k-5, k_2 = k-5, k_3 = k-3$, $k_4 = k - 6$ and $k_5 = k - 9$, from Subcase 2.2.

  Note that if reduction rule (2) of Phase 1 cannot be applied to $G - x$, then at least one the neighbors of u has degree 5, and so Sucacse 2.3 is impossible. In the degree 5 situation, four children are produced with parameter value $k - 5$. The total running time of the algorithm is $O(r^k k^2 + kn)$, $r = 4^{1/5}$, ou r =1.31951 approximately.

5.6 The Cheetham *et al.* Adapted algorithm

This algorithm is a combination of the Cheetham *et al.* Algorithm [6] and the Downey *et al.* Algorithm [13]. Note that both algorithms can solve the k-Vertex Cover problem, i.e they are not complementary and any one suffices to solve the problem. The proposed adapted algorithm combines the best parts of both, with the aim of improving the performance of the solution as a whole. While the algorithm of Cheetham *et al.* is in essence parallel, at the same time the parallelization of the algorithm of Downey *et al.* is not a trivial task and, indeed, the subject of future work.

The Cheetham *et al.* Adapted algorithm is described as follows. In the phase of reduction to problem kernel phase the algorithm of Cheetham *et al.*, based on Buss *et al.*. This algorithm receives as input data the instance (G_i, k_i) and performs the reduction to problem kernel in parallel. On the instance (G_i'', k_i'') resulting from the algorithm of reduction to problem kernel, we execute the algorithm B1 de Balasubramanian *et al.*, in the parallel version of Cheetham *et al*. The B1 algorithm computes a complete ternary search tree with height $\log_3 p$, where p is the number of processors. After this phase, each processor P_i possesses an instance (G_i''', k_i''') generated by a single path from the root to the leaves, i.e. each processor has a single instance of the problem, which was set from a number of different ramification choices in the descending path in the tree. Finally, each processor P_i exhaustively executes the sequential algorithm of Downey *et al* on the reduced instance (G_i''', k_i'''). Figure 2 shows how the Cheetham *et al.* Adapted algorithm works, changing to the B2 Algorithm for the Donwey Algorithm *et al.*.

In our implementation, we paid special attention to the operation of verifying rule 2 of the reduction to kernel phase, not only due to its high cost, but also its recurring nature in the execution of the algorithm. The control of the cost of this operation was fundamental to obtain the final good result. The operation of verification of rule 2 is to find a certain pair of non-adjacent vertices (u, v), such that $|N(u) \cup N(v)| > k$. It is not obvious how to find such non-adjacent vertices efficiently. To represent the graph G we used adjacency list. A naive algorithm, for example, could start at one vertex v and traverse its adjacency list to detect each non-adjacent vertex u to obtain the pair (u, v) and check the required property (i.e. $|N(u) \cup N(v)| > k$). But then all the non-adjacent pairs (u, v) of G would be tested, even those that do not have any chance to possess such property.

We need an algorithm to find such vertices quickly by testing *only* those non-adjacent vertex pairs that have some chance of possessing the required property. To this end, we first sort the pairs of vertices (u, v) in increasing order of the degrees. Then we test the adjacency condition and if the required property holds. In this way, the verification ends as soon as property $N(u) + N(v) \leq k$ is valid. To do this, we use a data structure call *Degree Controller*. It is a linked list of nodes called *Degree Nodes*. Each node contains a value of an existing degree in G and a pointer to a linked list of vertices of G that have that degree. The Degree Nodes are sorted by the degree value.

With this data structure we check the vertices starting from those with large degrees and proceed to smaller degrees, until we can stop testing. At each test, we check the non-adjacency condition and the required property. The tests are aided by the use of a bit vector of size $|G|$ in each vertex, with 1 in the bit vector denoting an adjacent vertex. In this way, to test the adjacency between u and v, we need only to check the bit vector, for example, of u to the position corresponding to v. To test the property $|N(u) \cup N(v)| > k$, it suffices to compute $|N(u) \bigcap N(v)|$ (quickly, also using the bit vector) and subtract it from $N(u) + N(v)$.

6 Experimental Results

In this section we present the experimental results of the FPT BSP/CGM Adapted Cheetham textitet al.

The computational environment is a *Cluster* consisting of 12 nodes: one node is AMD Athlon(tm) 1800+ 1GB RAM; one Intel(R) Pentium(R) 4 CPU 1.70GHz 1GB RAM; three Pentium IV 2.66GHz 512MB; one Pentium IV 2.8GHz 512MB; one Pentium IV 1.8GHz 480MB RAM; four AMD Athlon(tm) 1.66GHz 480MB RAM; AMD Sempron(tm) 2600+ 480 MB RAM. The nodes are connected by a 1Gb *fast-Ethernet switch*. Each node runs on Linux Fedora 6 with g++ 4.0 and MPI/LAM 7.1.2.

The input data used in our experiments are conflict graphs that represent sequences of amino acids collected by the NCBI database. This input is the same used by Cheetham *et al.* [6] and Hanashiro [16] in their experiments. Each value of the running time in the curves corresponds to the average time of 30 rounds. The times are in seconds and include the time to read de input data, to deallocate data structures and the time to write the output data.

We now show the performance results of running the Adapted Cheetham *et al.* algorithm, executed on 3 and 9 processors with input graphs Kinase, SH2 and Somatostatin. The values shown represent the average time of 30 experiments.

We emphasize here the use of adequate data structures and the backtracking technique in our implementation that have contributed to the good performance of the implemented algorithm, as shown in the next paragraphs. Implementing FPT algorithms it is not a simple task. We have tried using different data structures to achieve the presented

results. We have dedicated much effort to reduce the computational cost of Case 2 of the reduction to the problem kernel phase of the Algorithm of Downey *et al.*, one of the most costly operations in the algorithm.

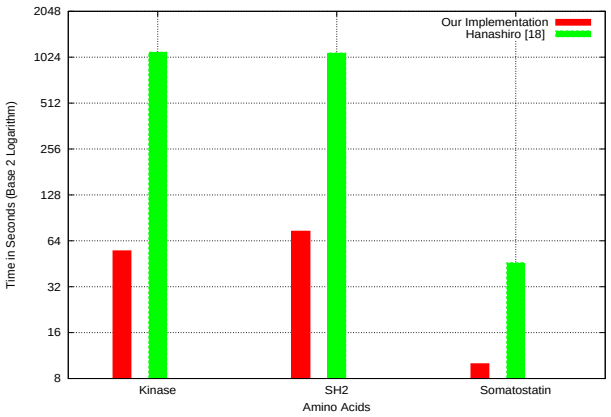

Figure 3. Average time obtained with Adapted Cheetham and Hanashiro [16] with 3 processors.

In figure 3, we compare the results obtained in our implementation versus the Hanashiro implementation, with 3 processors. We emphasize the good speedups obtained (relative to the other implementation) of 19.8 times for Kinase, 14.53 time for SH2 and 4.33 for Somatostatin.

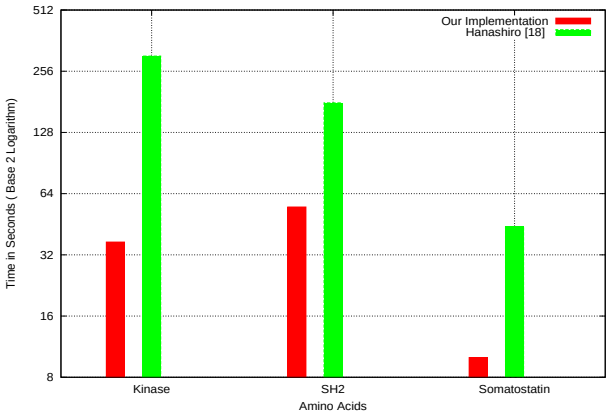

Figure 4. Average time obtained with the Adapted Cheetham and Hanashiro [16] using 9 processors.

In figure 4, we compare the results obtained in our implementation versus the Hanashiro implementation, with 9 processors. We again emphasize the good speedups (relative to the other algorithm) of 8.05 times for Kinase, 3.25 times for SH2 and 4.08 times for Somatostatin.

Grafo	3 processors	9 processors
Kinase	19,82	8,05
SH2	14,53	3,25
Somatostatin	4,33	4,08

Table 2. Speedups between the two implementations with 3 and 9 processors

In Table 2, we show the improvement of our proposed algorithm with respect to the Hanashiro implementation, with 3 and 9 processors.

For the PHD and WW graphs we do not obtain better results. It would be possible to develop an implementation that alternates the B2 and Downey et al. algorithms in the processors, in order to get good results for all the tested graphs. We noticed that, for the Kinase, SH2 and Somatostatin graphs, our implementation could reduce the instance of the problem more quickly, diminishing the search space in the search tree. This justifies the much smaller time spent to find the cover. Moreover, we observed that the improvement rate kept diminishing as we increased the number of processors. This is justifiable because with more processors, the space in the search tree is diminished in both implementations.

7 Conclusions and Future Works

FPT Algorithms have been successful in solving instances of NP-complete problems in practice for some important applications. The use of parallelism in FPT Algorithms has shown to be very useful and constitutes a further boost in performance. In the case of this paper, the proposed Adapted Algorithm of Cheetham *et al.*, produced an improved performance and substantial results were obtained.

We also consider that the use of good data structures and the backtracking technique have contributed to the good performance of the implementation. The implementation has gone through successive refinements until reaching this version. Implementing FPT algorithms it is not a simple task. Some versions, using different data structures have been implemented. We emphasize Case 2 of the reduction to the problem kernel phase of the Algorithm of Downey *et al.* as one of the most costly operations that we have dedicated particular effort to reduce its computational cost.

Among the works under development there are implementations of parallel FPT algorithms for the k-Vertex Cover problem that use other sequential algorithms. The implementations also will be executed in computational grid, as, for example, the InteGrade [15]. There are also FPT parallel algorithms being studied for other NP-Complete problems.

As a future work, it would be interesting to compare the presented algorithm with the one proposed by Abu-Khzam et al. [1]. We thank the anonymous referee for the suggestion.

Appendix

More formally, we state in the following the definition of parameterizable problem and fixed-parameter tractable according to Downey and Fellows [9].

Definition 2.1 A *parameterized language* L is a subset $L \subseteq \Sigma^* \times \Sigma^*$. if L is a parameterized language and $(x, y) \in L$ then we will refer to x as the *main part* and refer y as the *parameter* .

Definition 2.2 A parameterized language L is *fixed-parameter tractable* if it can be determined in time $f(k).n^{\alpha}$ whether $(x, y) \in L$, where $|x| = n$, α is a constant independent of both n and k and f is an arbitrary function. The family of fixed-parameter tractable parameterized languages is denoted *FPT*.

If a problem L is in the FPT class, then each associated problem L_y is solved in polynomial time by an algorithm whose exponent does not depend on the value of the parameter y.

To study and compare the complexity of parameterized problems Downey and Fellows [13] propose a notion of reduction of parameterizable problems that define a class hierarchy of parameterizable problems [13, 17].

The most successful techniques employed to design efficient fixed parameter tractable algorithms are *bounded search tree* and *reduction to problem kernel* [19]. The idea of the reduction to problem kernel, is to quickly solve some parts of the instance of the problem that are relatively easy to work with. The general idea of bounded search tree method is to identify a small subset of elements of which at least one must be in *any* feasible solution of problem.

Reduction to problem kernel. Let I be an instance of a parameterizable problem and a given parameter k. A reduction to problem kernel is an algorithm in polynomial time that transforms I into a new instance I' and k into a new parameter $k' \leq k$ that is independent of the size of the original problem I. The size of I' depends only on a function in k. Besides, the new instance I' has a solution regarding to k', if and only if the instance I has a solution regarding to the original parameter k.

Bounded search tree . The idea behind the phase of the Bounded search tree, is in how the search in trees is considered in relation to the depth of the search, that is limited by the parameter. Combined with some previous, but no obvious, knowledge from more efficient search mechanisms, the space for exploration of the searches becomes a lot smaller than those using a naïve brute force mechanism.

References

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: theory and experiments. L. Arge, G. F. Italiano, and R. Sedgewick (eds.). In *Proc. 6th Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmcs and Combinatorics (ALENEX-04)*, pages 62–69, 2004.

[2] C. E. R. Alves, E. N. Cáceres, and S. W. Song. Efficient representations of row-sorted 1-variant matrices for parallel string applications. In *Proceedings 7th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2007)*, volume 4494 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2007.

[3] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed-parameter algorithm for vertex cover. *Inf. Process. Lett.*, 65(3):163–168, 1998.

[4] J. F. Buss and J. Goldsmith. Nondeterminism within p. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[5] M. Cesati and M. D. Ianni. Parameterized parallel complexity. In *Euro-Par '98: Proceedings of the 4th International Euro-Par Conference on Parallel Processing*, pages 892–896, London, UK, 1998. Springer-Verlag.

[6] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large fpt problems on coarse-grained parallel machines. *J. Comput. Syst. Sci.*, 67(4):691–706, 2003.

[7] Chen, Kanj, and Jia. Vertex cover: Further observations and further improvements. *ALGORITHMS: Journal of Algorithms*, 41, 2001.

[8] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Proceedings of the ACM 9th Annual Computational Geometry*, pages 298–307, 1993.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1-2):109–131, 1995.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[11] R. G. Downey and M. R. Fellows. Parameterized complexity after (almost) 10 years: Review and open questions, 1999.

[12] R. G. Downey, M. R. Fellows, A. Vardy, and G. Whittle. Parameterized complexity: A framework for. systematically confronting computational. intractability. In *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99. Proceedings of the DIMACS-DIMATIA Workshop, 1999.

[13] R. G. Downey, M. R. Fellows, A. Vardy, and G. Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM J. Comput.*, 29(2):545–570, 1999.

[14] S. Gilmour and M. Dras. Kernelization as heuristic structure for the vertex cover problem. In *ANTS '06: Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence*, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[15] A. Goldchleger, F. Kon, A. G. vel Lejbman, and M. Finger. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. In *Proceedings of the ACM/IFIP/USENIX Middleware'2003 1st International Workshop on Middleware for Grid Computing*, pages 232–234, Rio de Janeiro, June 2003.

[16] E. J. Hanashiro. O problema da k-cobertura por vértices: uma implementação FPT no modelo BSP/CGM. mestrado, UFMS, Campo Grande, MS, Março 2004.

[17] Z. Lonc and M. Truszczynski. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Logic*, 4(1):91–119, 2003.

[18] Niedermeier and Rossmanith. Upper bounds for vertex cover further improved. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, 1999.

[19] R. Niedermeier. Some prospects for efficient fixed parameter algorithms. In *SOFSEM '98: Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics*, pages 168–185, London, UK, 1998. Springer-Verlag.

[20] U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical report, Aug. 10 1999.

[21] Valiant. A bridging model for parallel computation. *CACM: Communications of the ACM*, 33, 1990.