A New Genetic Convex Clustering Algorithm for Parallel Time Minimization with Large Communication Delays

J.E.P. Sanchez, D. Trystram

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 709-716, 2006.

http://www.fz-juelich.de/nic-series/volume33

A new Genetic Convex Clustering algorithm for parallel time minimization with large communication delays

J. Pecero Sanchez[a], D. Trystram[a]

[a]Laboratoire Informatique et Distribution (ID) IMAG, ZIRST 51 Av. Jean Kuntzmann, 38330 Monbonnot Saint Martin, France

1. Introduction

The efficient execution of an application on a parallel and distributed system highly depends on the decisions taken for scheduling the tasks that constitute the program. One solution for obtaining efficient parallel programs execution is task clustering. It corresponds to assign the tasks to clusters where each cluster is run on a single processor. For most of the new distributed platforms available today, communication delays are the main factor that affects the performance of these systems. Communications are usually relatively high in regard to basic execution time. The problem of clustering the tasks to be executed on a multiprocessor computer is one of the most challenging problems in parallel computing. There is no fully satisfactory results for the task clustering problem if large communication delay is considered.

We are interested here in the task clustering problem on an unbounded number of processors taking into account large communication delays. To solve it, we introduce a new genetic algorithm(GA) approach which has nice properties called Genetic Convex Clustering Algorithm (GCCA). The main idea is to assign tasks to locations in *convex* groups. We consider arbitrary execution time. We propose a novel crossover operator in the context of the task clustering problem.

The remainder of this paper is organized as follow. Section 2 presents the task clustering problem. An analisys of convex clusters properties will be presented briefly in Section 3. Section 4 introduce the GCCA, which considers the convex cluster properties. Section 5 evaluates the performance of the GCCA . Finally, Section 6 concludes this paper.

2. Task Clustering

Let us start by some preliminaries. As it is common, an application to be parallelized is represented as a precedence task graph. A precedence task graph is a weigthed acyclic digraph (DAG) *G=(V,E)*, where *V* is the set of tasks nodes, which are in one-to-one correspondance with the computational tasks in the application and *E* represents the set of the precedence relations between the tasks.

Clustering is often used as a compile-time preprocessing step in mapping parallel programs onto multipocessor architectures. In this context, given a precedence tasks graph and infinite number of fully connected processors, the objective of clustering is to assign the tasks to processors. The task clustering problem is *NP-Hard* [1][2], even when the number of processors is unbounded and tasks duplication is allowed. For most of the new parallel and distributed platforms available today, communication delays are the main factor that affects the performance. There is no fully satisfactory results for the task clustering problem if large communication delay is considered despite the result of Papadimitriou and Yannakakis [3], who proposed a constant approximation ratio when replication of tasks is allowed. If no replications are allowed, the problem of clustering with large communication delays has no satisfactory solution today. Thus, practical solutions have been proposed, based mainly

on three different heuristics: the algorithms based on the critical path analysis [1][4], priority-based list scheduling [5] and based on graph decomposition [6][7].

Another heuristic method used in the task clustering problem and scheduling context is the meta-heuristic known as Genetic Algorithm(GA) [8][9][10][14]. A genetic algorithm is a guided random search method where elements (called *individuals*) in a given set of solutions (called *population*) are randomly combined and modified (these combinations are called *crossover* and *mutation*) until some termination condition is achieved.

In this paper we introduce a new genetic algorithm approach which has nice properties called Genetic Convex Clustering Algorithm (GCCA). The main idea and the force of GCCA approach is to assign tasks to locations in convex groups. In the following sections we will detail this approach. First, we will analyze the convex cluster properties, after that the GCCA will be presented.

3. Analysis of Convex Cluster

In this section, we describe the convex cluster algorithm. The main idea is to assign tasks to locations in convex groups. We consider arbitrary execution time and large communication delays.

In the following, as introduced in Section 2, we consider a precedence tasks graph for representing the application to be parallelized. Let *G=(V, E)* denote such a graph. Let $\prec$ be the partial order of the tasks in G, its inverse relation $\not\prec$ and $\sim$ the equivalence relation defined as follows: for x and y $\in$ *V*, $x \sim y$ if, and only if $x \not\prec y$ and $y \not\prec x$. If two tasks x and y are such that $x \sim y$ we say that x and y are independent.

Definition 1 *A group of tasks* $T \subset V$ *is said* $convex$ *if and only if all pairs of tasks* $(x, z) \in T$*, all intermediate task y such that* $\forall$ *y,* $x \prec y \bigwedge y \prec z \Longrightarrow y \in T$ [7].

An example of convex cluster is depicted on Figure 1. A precedence task graph is depicted on left of Figure 1. Each node label shows the number of task and each edge label shows intertask communication cost between tasks. Each label beside of the nodes represents the task computation cost. The intra-communication cost (inside each cluster) on the clustered graph is zero.

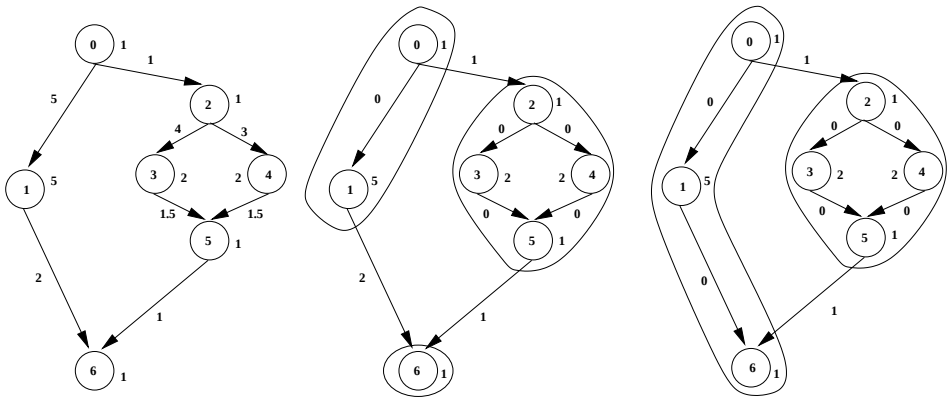

Figure 1. On left, a precedence task graph is depicted. In the middle, the graph is clustered on a valid convex cluster. On the right, we have a clustered graph but it is not convex.

The convex clustering has the following properties:

Property 1. The convex cluster is a particular class of $Processor\ Ordered\ Schedule$ [11], it means, if a cluster is assigned to processor i, the predecessors of all tasks belonging to this cluster

are into the processors with index less than i.

Proof. The proof is by contradiction. Let assume that a cluster C_i is not convex and is assigned to processor with index i, then there exist *x* and *z* $\in C_i$ and $y \in C_j$ assigned to processor with index $i+1$ such that $x \prec y \prec z$. As $x \prec y$, there exists a path between processors i and $i+1$, and similarly there exists a path between processors $i+1$ and i because $y \prec z$. It is impossible by definition of convex cluster and by definition of processor ordered schedule.

Proposition 2. The resulting clustered graph in a convex cluster is a Directed Acyclic Graph.

Proposition 3. This class of algorithm with the convex cluster properties and taking into account unit execution time and large communication delays was proved to be *2-Dominant*, that is, there exists a convex clustering whose execution time on an unbounded number of processors is less than twice the time of an optimal clustering.

So, this makes a good solution for solving practical scheduling algorithms since we expect to be able to schedule more easily convex clusters than general graphs.

Now that we have shown the strength of convex clustering, it remains to find a "good" convex clustering. For this problem in [7] proposed an algorithm based on a recursive decomposition of the precedence task graph. This algorithm does not allow to obtain all the convex clusters. The problem of convex cluster based on a recursive decomposition is NP-Complet [13]. So, to build all the convex cluster we propose to use genetic algorithm.

4. Genetic Convex Clustering Algorithm

Genetic Algorithm (GA) is a guided random search algorithm based on the principles of evolution and natural genetics. It combines the exploitation of past results with the exploration of new areas of search space. By using survival of the fittest techniques and a structured yet randomized information exchange, genetic algorithm can mimic some of the innovative flair of human search. Genetic algorithm is randomized but not simple random walks. It exploits historical information efficiently to speculate on new search points with expected improvement [12].

The research on using GAs for clustering and scheduling tasks in parallel computing is an active research, with many papers showing the potential use of this class of algorithms [10][14]. Each approach contains its own characteristics, but the main difference of these algorithms is the representation of a solution. In this section, we describe our proposed approach.

4.1. Algorithm design

The Grouping Genetic Algorithm is a genetic algorithm heavily modified to suit the structure of grouping problems. Those are the problems where the aim is to find a good partition of a set, or to group together the members of the set [15]. GCCA belong to this class of algorithm. To represent a valid solution and to handle the genetical operators (i.e crossover) with convex cluster, we proposed the following representation.

Coding

A solution representation of a task clustering problem based on convex cluster is feasible if it satisfies the following conditions:

1. Each task of the precedence tasks graph belongs to exactly one cluster.
2. The restriction of convexity must be fulfilled.
3. The fitness function of GCCA depends only on the clustering of the tasks, rather than the numbering of the cluster.

To represent a valid convex cluster, we propose a bichromosomal representation. The first chro-

mosome encoded the number of tasks and the second chromosome encoded the number of groups. We augmented this representation with a group part, that is, the group part encoding the cluster on a one task for one cluster basis. The length of the bichromosomal representation is equal to the number of tasks and the length of the group part representation is variable and this one depends on the numbers of clusters. The Figure 2 depicts a valid solution representation, meaning the tasks 0 and 1 are assigned in cluster 0, the tasks 2, 3, 4 and 5 in cluster 2 and finally the task 6 is assigned to processor 3. The group part of the chromosome represents only the groups(clusters), in this case, express the fact that there are three clusters in the solution. The main fact is that the genetic operators will work with the group part of the chromosomes, the standard task part of the chromosomes identify which items actually form which group.

Figure 2. A chromosome representation of the convex clustered graph depicted in the middle of the figure 1.

Initial population

The first step in GCCA is to create an initial population. Most of the genetic algorithms applied to the task clustering problem create a random initial population. We propose an algorithm to generate the initial population. This algorithm was adapted to build feasible convex clusters. The algorithm is the following:

1. Put all the tasks in a list called ListTasks.
2. Choose randomly a task of ListTask and assign it a cluster.
3. Compute the predecessors set and the successors set of the selected task.
4. Eliminate these tasks of the ListTasks.
5. **While** (ListTasks != $\emptyset$)

5.1 Choose randomly an independent task and to assign it a different cluster.

a). Compute the predecessors set and the successors set of the selected task.

b). Eliminate these tasks of the ListTasks.

EndWhile

6. Copy the predecessor sets in a global predecessor set.
7. Assign into a cluster all the tasks that are duplicated in the global predecessor set.
8. Eliminate all the duplicated tasks of the corresponding set.
9. Copy the successors sets in a global successor set.
10. Eliminate all the duplicated tasks of the corresponding set.
11. Assign the remaining tasks of the predecessor and successor sets in the correspondant cluster.

Evaluation

The genetic algorithm aim is to maximise the fitness function, while minimising the objective function. To evaluate the fitness of each solution, first the schedule length of the particular solution is determined and after that, this schedule length is mapped to an initial fitness value (f_i') using equation (1) [9]:

$$f_i' = sumofallthetasksexecutiontimes - schedulelength. \quad (1)$$

Where the *sum of all the tasks execution times* is defined as the time it would take to run all the tasks in the DAG on a single processor. However, in some cases f_i' can still be negative since some of the initial clusterings can result in schedules whose length is worse than running the tasks on a single processor. So a linear scaling is employed to obtain the final fitness value (equation (2)).

$$f_i = \frac{f_i' - MIN[f_i']}{MAX[f_i'] - MIN[f_i']} \tag{2}$$

To obtain the schedule length for a particular clustering each task is scheduled in decreasing order priority. That is, schedule a task into the first slot available after the specified earliest start time on the assigned processor. The earliest start time $e(v)$ for task v is calculated using the equation (3):

$$e(v) = MAX[s(u) + \mu(u) + \lambda(u,v), (u,v) \in E] \tag{3}$$

Where: $s(u)$ is the starting execution time of task u, $\mu(u)$ represents the execution time of task u, $\lambda(u,v)$ represents the tasks communication time. $\lambda(u,v) = 0$ if tasks u and v are placed on the same processor. Once all tasks have been scheduled, the algorithm uses the equation (4) to determine the schedule length of the scheduled DAG.

$$MAX[s(v) + \mu(v)] \tag{4}$$

Stop condition

The obvious way to stop the genetic algorithm would be to wait until the optimum is found, but due to the complex solution spaces there is no indication whether the optimal or only a near-optimal solution has been located. So, there are three ways GCCA will terminate. Firstly, if it has reached the maximum number of iterations. Secondly, when the best fitness in a population has not changed for a specified number of generations. Finally, when the difference between the average and the best fitness remains constant for some given number of generations.

4.2. Genetic operators

Selection

Selection uses a proportionate selection scheme and replaces the entire previous generation. The proportionate selection scheme where a string with fitness value f_i is allocated to a relative fitness of f_i/f_{aver}, where f_{aver} is the average fitness of the population. GCCA uses the *roulette wheel* style of selection to implement proportional selection. The algorithm selects strings until the next generation is completely generated.

Crossover

Crossover produces new individuals that have some portions of both parent's genetic material. As pointed out in the previous section, GCCA crossover will work with variable length chromosomes with genes(group part) representing the clusters. To generate a valid solution we used the following algorithm:

1. Select randomly two crossing sites, delimiting the *crossing section*, in each of the two parents.
2. Inject the contents of the crossing section of the first parent in the first crossing site of the second parent. Recall that a crossover works with the group part of the chromosome, so this means injecting some of the *clusters* from the first parent into the second.
3. Eliminate all *tasks* now occurring twice from the clusters they were members of in the second parent. Consequently, some of the old groups coming from the second parent are altered: they do not contain all the same tasks anymore, since some of those tasks had to be eliminated.

4. If necessary, *adapt* the resulting clusters, according to the convex cluster constraints.

5. Apply the points 2. through 4. to the two parents with their roles permuted in order to generate the second child.

Considering the graph depicted in Figure 1, a crossover example is depicted in Figure 3.

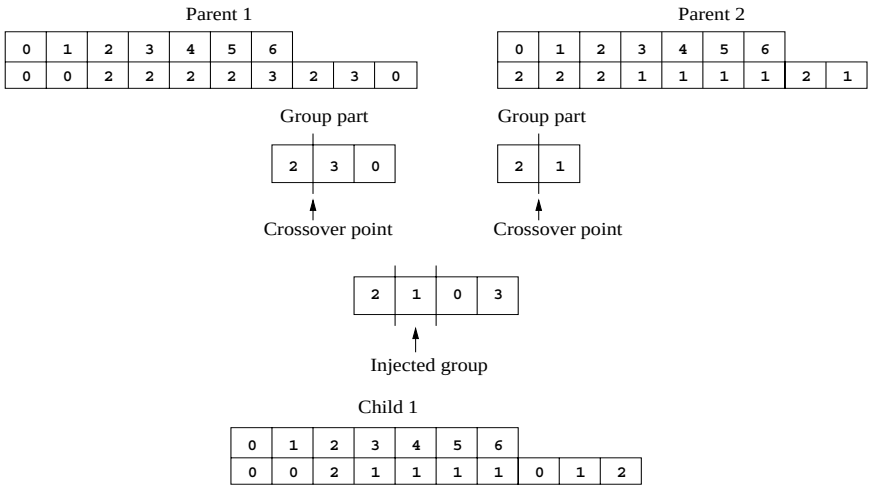

Figure 3. A crossover example.

4.3. Mutation

The main objective of mutation operation is to produce a slight perturbation in the search, in order to quit a local minimum. The GCCA mutation works by changing a task's cluster (*if it is possible*) to any cluster other than the one it was originally.

5. Results

In producing the results given in this section, the following benchmark values were adopted: population size of 100, crossover probability of 0.8, mutation probability of 0.01, a generation limit of 100, unlimited number of processors.

To evaluate the performance results of GCCA, we used a benchmark of *Reference Graphs* (RG). The Reference Graphs are tasks graphs that have been previously used by different researchers and addressed in the literature. This set consists in about 9 graphs (7 to 18 task nodes). These graphs are relatively small graphs but do not have trivial solutions and expose the complexity of clustering very adequately.

Figure 4 shows the performance of GCCA in a graph of 10 tasks. The task graph used here has an optimal schedule length of 26 unit times and 2 clusters [9]. On left of Figure 4, the convex clustering obtained by GCCA is showed. In the middle, the Gantt chart (a graphical representation of the duration of tasks against the progression of time) represents the best solution performed by GCCA. On the right, the Gantt chart depicts the optimal schedule reported in the literature. GCCA obtains the optimal schedule length but the number of processors is greater than the number reported in [9] which has only 2. It is due to the fact that GCCA tries to use all the available processors.

Table 1 summarizes the experiment results obtained by GCCA. The first column shows the reference of the graphs. The number of tasks in the RG are shown in the second column. Third and fourth

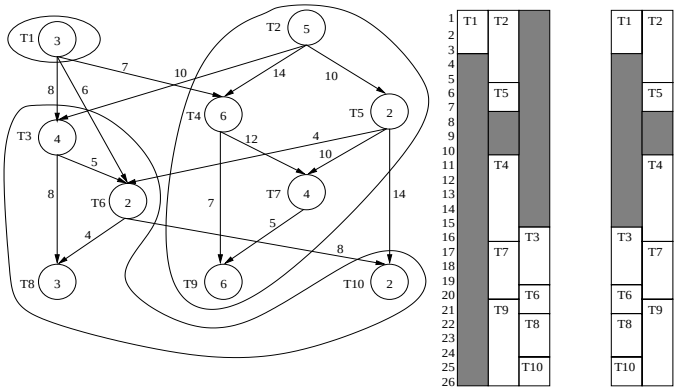

Figure 4. Convex cluster solution to the problem of 10 tasks. The tasks are represented in white, idle times are represented in gray.

References	No. nodes	Optimal schedule	Optimal clusters	GCCA schedule	GCCA clusters
[9]	7	9	2	10	3
[18]	7	4	2	4	4
[9]	9	5	2	5	5
[20]	9	149	2	160	3
[9]	10	26	2	26	3
[19]	10	11	2	10	6
[17]	13	190	2	190	5
[16]	18	440	3	480	5
[16]	18	330	3	340	6

Table 1
Table of results.

columns represent the optimal schedule length and the optimal number of clusters reported in the literature, respectively. Finally, the best schedule and the best number of cluster obtained by GCCA are reported in the two last columns of the figure. The experiment results show that GCCA is able to find optimal solutions for some test cases. In some other cases the number of clusters computed by GCCA is greater than the optimal number of clusters reported in the literature.

6. Conclusions

In this work we presented a new genetic algorithm called Genetic Convex Cluster Algorithm, which has nice properties. This algorithm was applied to solve the task clustering problem with large communication delays. The experiment results obtained by GCCA show the feasibility of using genetic algorithm with the convex cluster properties to solve the task clustering problem. The GCCA approach seems particularly well suited for the new parallel systems like cluster of PC with hierarchical communications. We proposed two novel genetic operators (representation and crossover) in the context of the task clustering problem.

References

[1] V. SARKAR (1989). *Partitioning and Scheduling Parallel Programs for multiprocessors.* Pithman, 1989.

[2] P. CHRÉTIENNE AND E.G.COFFMAN AND J.K.LENSTRA AND Z.LIU (1985). *Scheduling Theory and its Applications*, chapter *Scheduling with Communications Delays: A Survey.* John Wiley and Sons (1985), New York.

[3] C. H. PAPADIMITRIOU AND M. YANNAKAKIS (1990). *Towards and architecture independent analysis of parallel algorithms*. SIAM Journal on Computing, vol. 19. pg. 322-328.

[4] A. GERASOULIS AND T. YANG (1993). *DSC: Scheduling parallel tasks on an unbounded number of processors*. IEEE Transaction on Parallel and Distributed Systems, vol. 4, num. 6, pag. 686-701.

[5] J.J. HWANG AND Y. CH. CHOW AND F. D. ANGERS AND CH. Y. LEE (1989). *Scheduling precedence graphs in systems with Interprocessor communication times*. SIAM J. Comput., vol. 18, num. 2, pp. 244-257, April.

[6] Y. K. KWOK AND I. AHMAD (1999). *Static scheduling for task graph allocation*. ACM Computing Surveys, vol. 31, num. 4, December.

[7] R. LEPÈRE AND D. TRYSTRAM (2002). *A new clustering algorithm for scheduling with large communication delays*. 16th IEEE-ACM annual International Parallel and Distributed Processing Symposium (IPDPS'02). April 15-19. Marriott Marina, Fort Lauderdale, Florida

[8] E. HART, P. ROSS, D. CORNE (2005). *Evolutionary scheduling: a review*. Genetic Programming and Evolve Machines, vol. 6, pag. 191-220. 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

[9] A. Y. ZOMAYA AND G. CHAN (2004). *Efficient clustering for parallel tasks execution in distributed systems*. 18th IEEE-ACM annuals International Parallel and Distributed Processing Symposium (IPDPS'04). April 26-30. Santa Fe, New Mexico, Eldorado Hotel.

[10] A. S. WU, H. YU, S. JIN, K. LIN AND G. SCHIAVONE (2004). *An incremental genetic algorithm approach to multiprocessor scheduling*. IEEE Transactions on parallel and distributed systems, vol. 15, num. 9, September.

[11] F. GUINAND, A. MOUKRIM, AND E. SANLAVILLE (2004) *Sensitivity analysis of tree scheduling on two machines with communication delays*. Parallel Computing, 30(1), pages 103-120, 2004.

[12] D. GOLDBERG (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, Mass.: Addison-Wesley.

[13] B. GAUJAL, G. HUARD, E. THIERRY AND D. TRYSTRAM (2004). *Convex Clustering*. 1st Bertinoro Workshop on Algorithms for Scheduling and Communication. 27 june - 3 july 2004. University of Bologna Residential Center. Bertinoro, Italy.

[14] R. C. CORREA, A. FERREIRA, P. REBREYEND (1999). *Scheduling multiprocessor tasks with genetic algorithm*. IEEE Transactions on parallel and distributed systems, vol. 10, num. 8, August.

[15] EMANUEL FALKENAUER (1996). *A hybrid grouping genetic algorithm for bin packing*. Journal of Heuristic, vol. 2, num. 1, pag. 5-30.

[16] Y. KWOK (1994). *Efficient Algorithms for Scheduling and Mapping of Parallel Programs onto Parallel Architectures*. M.Phil. Thesis, Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, June, 1994. url = "citeseer.ist.psu.edu/kwok94efficient.html

[17] J.D. EVANS, R.R. KESSLER (1992). *A communication-Ordered Task Graph Allocation Algorithm*. IEEE Transactions on Parallel and Distributed Systems, 1992.

[18] Y.-K. KWOK, I. AHMAD (1999). *Static scheduling algorithms for allocating directed taskgraphs to multiprocessors*. ACM Computing Surveys, vol. 31 num. 4, december, pag. 406-471. 1999.

[19] MIN-YOU WU, WEI SHU, JUN GU (2001). *Efficient Local Search for DAG Scheduling*. IEEE Transactions on Parallel and Distributed Systems, vol. 12 num. 6, june, pag. 617-627. 2001 IEEE Press.

[20] C. L. MCCREARY, M. A. CLEVELAND, AND A. A. KHAN (1996). *The Problem with Critical Path Scheduling Algorithms*. Technical Report 9601, Auburn University. Department of Computer Science and Engineering. January 2, 1996.