*Multi-Level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning

Michel Toulouse[1] Krishnaiyan Thulasiraman[2] Fred Glover[3]

[1] Department of Computer Science
University of Manitoba
[2] School of Computer Science
University of Oklahoma
[3] Graduate School of Business
University of Colorado

Abstract. Cooperative search is a parallelization strategy for search algorithms where parallelism is obtained by concurrently executing several search programs. The solution space is implicitly decomposed according to the search strategy of each program. The programs cooperate by exchanging information on previously explored regions of the solution space. In this paper we propose a new design for cooperative search algorithms which is also a new parallel problem solving paradigm for combinatorial optimization problems. Our new design is based on an innovative approach to decompose the solution space which is inspired from the modeling of cooperative algorithms based on dynamical systems theory. Our design also gives a new purpose to the sharing of information among cooperating tasks based on principles borrowed from scatter search evolutionary algorithms. We have applied this paradigm to the graph partitioning problem. We describe the parallel implementation of this algorithm on a cluster of workstations and compare our results with other well known graph partitioning methods.

1 Introduction

Given that a finite amount of computational time is allocated to find a near optimal solution, local search methods often stand to benefit from parallelization to find better solutions. It is common to base the parallelization on *restarts* of the local search from different initial points in the solution space and/or to use different search criteria to improve the quality of the solutions returned by the search method. Restarting a search method yields an *implicit decomposition* of the solution space, and if these restarts do not require knowledge of the outcome of other restarts, we can exploit them by creating independent tasks for a concurrent exploration of the solution space using parallel computers. Following the principle underscored in tabu search that the history of the solution process

* P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 533-542, 1999.

is important for determining decisions made at various stages, it has been observed that the search methods can benefit from information gathered by the other concurrent search threads in a parallel search. This has given rise to a parallelization paradigm called *cooperative search* (Toulouse, Crainic and Sansó 1997).

In this paper we propose an innovative design for cooperative search algorithms which is also a new parallel problem solving paradigm for combinatorial optimization problems. Current design of cooperative search algorithms are mostly based on search programs working with the same data set and exploring the solution space according to an implicit partition obtained from different search strategies. In our proposed design, each search program works with a different data set, each object of each data set is a partial solution of the optimization problem and each partial solution of a data set is obtained from objects of the previous data set in the hierarchy and also serves as a building block of objects in the next data sets in the hierarchy. The second aspect in which our design departs from current cooperative algorithms is relative to the sharing of information. In our new design, shared information is used to reorganize the partial solutions in the data sets. Given a particular instance of a hierarchy of data sets, the exploration of the problem solution space is limited to regions that can be reached by the neighborhood functions defined by the set of partial solutions at each level. By reorganizing the partial solutions in the data sets, i.e. destroying some partial solutions and creating others, new data sets (and consequently new neighborhood functions) are created at each level where such a reorganization occurs, which provides for new regions of the solution space to be explored. This strategy uses principles borrowed from *scatter search* evolutionary algorithms (Glover 1977) and *vocabulary building* procedures (Glover 1992, Glover and Laguna 1993, Glover and Laguna 1997) to guide the "combination" of shared information originating from different sources.

We have applied this problem solving method and cooperative search design to the k-way graph partitioning problem (GPP). In terms of the quality of the solutions, our approach turns out to be by far the most powerful heuristic for this optimization problem. The performance of this approach is also consistent, always outperforming the other algorithms for all the problem instances tested. This outcome demonstrates that the present design is highly successful in adapting the dynamical behavior of the system to the optimization objective of the problem.

The rest of paper is organized as follows. In Section 2 we introduce a formalization of the GPP and classify the existing graph partitioning methods. In Section 3 we introduce our multi-level cooperative search framework. In Section 4 we describe an application of our multi-level cooperative search to the GPP. In Section 5 we give the resulting edge cuts from a parallel implementation of the algorithm to several graphs. Concluding remarks are provided in Section 6.

2 The multi-level cooperative search framework

Cooperative algorithms are relatively new, and most of the attention thus far has focused on finding useful mechanisms for sharing information to improve the search behavior of cooperating programs. However, cooperation issues are more subtle than simply considering the impact of reused information on the performance of cooperating programs. Shared information among the search programs creates an inextricable network of dependencies which are most likely to be at the source of the divergent behavior of cooperating algorithms when compared to the individual search strategies that compose them. Our concern is to organize the dependencies among cooperating programs in a more manageable control structure in terms of how it affects the search of cooperative algorithms.

Similar issues have been addressed previously by scatter search evolutionary methods and their derivatives, notably the vocabulary building procedures. Scatter search approaches are population based search heuristics that seek to exploit information contained in "elite solutions" by combining them to derive new solutions. The role of the elite solutions in scatter search as a means to control the exploration of the solution space is based on strategies whose roots can be traced in part to OR and AI proposals at the end of the 50s and early 60s. (For a review see Glover 1997 and Glover and Laguna 1997.) Vocabulary building constitutes a variation of scatter search which focuses on components of elite solution vectors (partial solutions) rather than complete solutions. Procedures using this design have been implemented in a sequential programming setting by Rochat and Taillard (1995), Kelly and Xu (1995), Taillard et al. (1995) and Lopez, Carter and Gendreau (1996). Elite solutions are identified using local search methods. Partial solutions are extracted from the elite solutions to form a pool of *solution fragments*, which in turn are combined to form larger fragments until complete solutions are generated.

By adapting the data sets of the search programs to embody a hierarchical structure of the form proposed above, we have been able to convert current cooperative algorithms into multi-level structures. Our design is based on assuming that solutions of the combinatorial optimization problem can be represented as sequences or partitions. Decision variables that define the objective function serve as the logical choice to constitute the lowest level components of the multi-level structure, and we progressively combine these variables to provide larger aggregates of variables. Thus the original decision variables are building block-s for the second level of the structure, whose elements are larger sequences or partitions of solutions. These larger aggregates again function as building block-s, to be combined into higher order assemblies, forming a set of hierarchically structured complexes.

A large number of multi-level structures can be obtained by combining the decision variables in different ways – a choice that is available at each level of aggregation. Each combination strategy defines an *instance* of a multi-level structure that organizes the original data set of the problem. Each set of aggregates of a given instance of this structure, which provides decision criteria for all levels, is used as an input (a "data set") for one of the search programs

composing a cooperative algorithm.

If we use only one multi-level structure instance (as in an effort to design multi-level graph partitioning algorithms), higher level controls may not be able to give very favorable guidelines to lower control levels, and thus may yield a poor exploration of the solution space. A better approach is to explore the solution space according to different instances of the multi-level structure. Instances can be generated statically (using different combination strategies) with or without the knowledge of the results of the exploration of previous instances, or they can be generated dynamically. We have favored the latter option in our design. We generated instances dynamically by using interactions occurring among the levels being explored by the search programs. In our current design we have defined three different kinds of local interactions which are implemented in terms of inter-task operators: an *interpolation operator*, a *destroy operator* and a *create operator*. As in other cooperative algorithms, these local interactions have a cumulative impact and give rise to diffusion processes among the programs of the multi-level structure. However, unlike previous designs, the local interactions are part of a hierarchical control structure which evolves according to the optimization logic of the search methods. This helps to create dynamics in the system favoring a better exploration of the solution space of the optimization problem.

3 Application to the graph partitioning problem

The graph partitioning problem is a combinatorial optimization problem of practical interest in many engineering fields. Given a graph $G = \{V, E\}$ where $|V| = n$ and $|E| = m$, the graph partitioning problem seeks to partition the set of vertices V into k subsets $V_1, V_2, \ldots, V_k$ in order to minimize

$$\sum_{i=1}^{i=m} w(e_i) \mid e_i \text{ connects vertices belonging to different subsets}$$

subject to $V_i \bigcap V_j = \emptyset \; (i \neq j)$; $|V_i| - |V_j| \leq 1$, $\forall\, i, j \in \{1, \ldots, k\}$; and $\bigcup_{i=1}^{i=k} V_i = V$. In this section we introduce a parallel implementation of a cooperative algorithm for the graph partitioning problem which incorporates the design principles described in the previous section. First we describe the initialization phase of the procedure.

3.1 Initialization

The initial set of components is based on the decision variables of the optimization problem. For the graph partitioning problem, this initial set corresponds to the vertices of the graph. To obtain the components (aggregates) at the second level of the multi-level data set to exploit the building block effect, we aggregate the initial set of components into partial solutions. For the graph partitioning

problem, this consists of creating aggregates of vertices which become the components to be manipulated at the second level of the structure.

In our current implementation of the multi-level cooperative algorithm for the GPP, we have a single hierarchy of data sets. These data sets are obtained using a maximal matching algorithm (Hendrickson and Leland 1993 and Karypis and Kumar 1998) which consists of finding a maximal set of edges in the graph such that no two edges are incident to the same vertex. (The term "maximal", in contrast to "maximum", refers to a matching that is locally optimal in a constructive sense, i.e., it is contained in no larger matching.) The two vertices i and j of an edge (i, j) in the matching are merged into a single vertex. The new vertex weight equals the sum of the weights of its constituent vertices and the vertex also retains the edges adjacent to its constituent vertices. Edge weights are left unchanged, except where vertices i and j both are joined by edges to a common vertex k before being merged. Then the two edges (i, k) and (j, k) are replaced by a single edge whose weight equals the sum of the weights of (i, k) and (j, k).

Denote graph $G = \{V, E\}$ by $G_0 = \{V_0, E_0\}$. The aggregates built from the initial set of vertices V_0 using the maximal matching algorithm defines a new graph $G_1 = \{V_1, E_1\}$ where $|V_1| \approx |V_0|/2$. The maximal matching is then applied to graph G_1, and iteratively to the graph obtained from G_1 until the coarsened graphs are small enough to be easy to explore. Assuming that p graphs have been generated by iteratively applying the maximal matching algorithm to the last generated graph, we then have after $p - 1$ aggregation operations, p graphs $G_0, G_1, \ldots, G_{p-1}$.

3.2 The control structure

The control structure represents the problem solving method. In this multi-level algorithm there are two types of control structure (two types of neighborhood functions). The first one consists of the graph partitioning algorithms applied to the data set at each level (the traditional neighborhood function defined by the search move used, swap, 2-opt, 3-opt, etc.). Essentially we use constructive or local improvement partitioning algorithms to search the graphs at each level of the structure, therefore performing the search of all the levels in parallel.

The interactions among the data sets at different levels constitute the second type of control structure. While the search space of the levels $i > 0$ is increasingly smaller and easier to search, the search processes at these levels can only explore partially the solution space of the problem and may not yield very good solutions. We need to create new multi-level structure instances. This is achieved through interactions among consecutive levels of the structure, changing the sets of aggregates. At this point we elaborate the three operators which are used to implement the interactions among the cooperating programs of the multi-level structure.

Destroy operator: It determines whether the vertices of an aggregate in G_i lie in the same set of the best partition in graph G_{i-1}. If not, destroy the aggregate in graph G_i so that each of the vertices of the aggregate constitute a separate set.

Each process P_i has a data structure which records which vertices of the graph G_i have been merged together in a single vertex (aggregate) of graph G_{i+1}. To implement the destroy operator, processes $P_i, i = 0, \ldots, p-2$ compare a good partition of graph G_i with the set of aggregates in the graph G_{i+1}. If an aggregate in graph G_{i+1} has components (vertices) from graph G_i which are not in the same set in the good partition, this aggregate is marked to be destroyed. This step is followed by a communication phase where process P_i sends the set of marked aggregates to process P_{i+1}. Once a process $P_i, i = 1, \ldots p-1$ has received the information about which aggregates have to be destroyed, it proceeds to remove these aggregates from the graph G_i and to create a new set of vertices in graph G_i from the components of the destroyed aggregates. The outcome of executing the destroy procedure is a new multi-level structure instance consisting of modified data sets potentially at each level of the structure. These data sets yield new graphs in the sequence of graphs (except for graph G_0). The new graphs are based on different sets of partial solutions, therefore providing a new set of neighborhood functions to search the graph G_0. Processes need then to resume the exploration of the solution space of G_0.

Create operator: Given vertices which are often in the same set relative to a collection of good partitions in graph G_i, create an aggregate with these vertices and make it a new aggregate (vertex) of graph G_{i+1}.

The sequence of operations of the procedure to implement the "create" operator is similar to the sequence for the "destroy" operator. New aggregates for graph G_i are identified based on elite solutions found when process P_i explores graph G_i. A variety of strategies and guidelines have been proposed in the literature on scatter search methods (and their path relinking generalizations) to combine such elite solutions. In our current implementation, when a good solution is visited, we record in which set of the partition each vertex is found. Then we use a simple frequency memory to identify "consistent variables" following a scatter search theme that is shared with tabu search. Specifically, before applying the create operator, those vertices which have often been together in the same set are considered candidates to be merged in a single vertex of graph G_{i+1}, i.e. to become the building blocks or components of new aggregates in graph G_{i+1}. Following this initial phase of the "create" procedure, the sequence of operations of the "create" operator is identical to the sequence for the "destroy" operator.

Interpolation operator: The current best partition in G_{i+1} is transformed by this operator to become an initial solution for the exploration of graph G_i. In particular, the operator places nodes of G_i in the same sets of the partition as their aggregate in the best partition in G_{i+1}.

The procedure that implements the "interpolation" operator for process P_i, receives a good partition of graph G_{i+1} from process P_{i+1}. Assume aggregate a of graph G_{i+1} is in the set A_{i+1} of the good partition received from process P_{i+1}. The aggregate a is then uncoarsened and all components of the aggregate a are placed in the set A_i of the partition for graph G_i. This yields an initial solution for process P_i from which a local search is launched to explore the solution space of graph G_i.

3.3 The parallel implementation

In our current implementation, these three operators are in sequence in the main loop of the program. The main loop iterates in a synchronized manner, applying first the destroy operator at all levels in parallel, followed by the interpolation and create operators again at all levels, before entering in the next iteration. The implementation and experiments described in this paper have been run on a cluster of Sparc workstations. The multi-level structure is a sequence of graphs. If the multi-level structure has p levels, we reserve a set of p processors and define an interconnection topology among them corresponding to an array. Let processors P_{i-1} and P_{i+1} be the neighbors of processor P_i in the array topology. This means that our communication software need to send messages only to processors P_{i-1} and P_{i+1} from processor P_i. In this array topology, P_{i-1} is not defined for P_0 and P_{i+1} is not defined for P_{p-1}. Our mapping strategy is to associate the process which searches graph G_i to processor P_i in the array such that levels G_{i-1}, G_i and G_{i+1} are neighbor tasks in the logical array configuration of processors. All data exchanges among the processes use our own communication software, which is stream based using the TLI protocol interface. (We have also realized an implementation of this multi-level cooperative search under PVM, and a second one that runs on the Origin 2000 shared memory parallel computer.)

This parallel algorithm is an implementation of the framework described in section 2. We believe that similar implementations can be realized for several optimization problems where solutions can be expressed as a combination of partial solutions.

4 Experiments

The experimental results show that with only a summary implementation, our new method is actually quite effective. The tests have been performed using the set of graphs in Table 1 as benchmark set (which represents the complete set of graphs that we have tested).

Graphs	# of vertices	# of edges	Graphs	# of vertices	# of edges
3elt	4720	13722	tooth	78136	452591
whitaker3	9800	28989	rotor	99617	662431
4elt2	11143	32818	ocean	143437	409593
4elt	15606	45878	auto(ef_589)	110971	741934
sphere	16386	49152	m144	144649	1074393
bcsstk31	35588	572914	m14	214765	1679018
brack2	62631	366559			

Table 1. Characteristics of the graphs in the benchmark set

These graphs have been widely used as benchmarks for graph partitioning algorithms. Most of these graphs are triangular and quadrangular unstructured meshes related to fluid dynamics, structural mechanics, or combinatorial optimization problems, obtained from the web site:

$ftp://ftp.u-bordeaux.fr/pub/Local/Info/Software/Scotch/Graphs$

Graphs	Methods	Number of sets in the partitions / Edge cut								
		2	4	8	16	32	64	128	256	512
3elt	C2.0	91	225	386	629	1102	1736			
	M3.0	90	221	386	631	1098	1723			
	MO	90	205	347	578	818	1649			
whit	C2.0	132	392	700	1218	1879	2814			
	M3.0	131	427	716	1219	1886	2811-1			
	MO	128	382	667	1131	1754-1	2661-1			
4elt2	C2.0	130	351	656	1154	1809	2814			
	M3.0	130	359	658	1122	1784	2830-2			
	MO	130	351	617	1048	1710-1	2632-1			
4elt	C2.0	147	384	647	1111	1880	2994			
	M3.0	142	380	645	1043	1738	2938			
	MO	139	337	557	1038	1665	2726			
sphere	C2.0	430	824	1273	2016	2893	4144			
	M3.0	424	864	1333	2059	2943	4183			
	MO	386	776	1202	1847	2695-1	3924-2			
btk31	C2.0	3396	9423	17262	28656	45291	67134			
	M3.0	2815	7879	14426	26366	43665	67533-1			
	MO	2795	7736	14356	25175	40264-1	61563-1			
brack2	C2.0	734	3365	7851	13126	19975	29725	42472	60215	83498
	M3.0	742	3426	7753	13289	19940	29409	42560-30	59036-14	81001-8
	MO	731	3143	7669	12109	18412-1	27790	41061-1	57472-1	79429-1
tooth	C2.0	4307	7940	13332	20061	29282	40417	54186	72593	96930
	M3.0	4618	7854	13329	21249	29061-1	40260-1	53224-38	71256-19	94355-11
	MO	3877	7241	12580	19238	27243	37427	50583-1	67929-1	91472-1
ocean	C2.0	468	2089	5126	9496	15207	23138	32110	43904	59122
	M3.0	509	2054	5114	9079	15614	23745-1	34630-33	46848-34	61096-28
	MO	472	1928	4468	9057	14439	22212-1	31185	42715-1	58169-1
rotor	C2.0	2157	8300	15108	24171	37546	54101	75373	104949	142131
	M3.0	2647	8354	15359	24841-1	36860-1	52626	74283-50	100063-24	134213-14
	MO	2113	7762	13686-1	21720	33585-1	49243-1	70084-1	96960-1	131490-1
auto	C2.0	2486	8855	17925	30885	45261	64576	89532	120717	159149
	M3.0	2444	8631	18028	29268	45716	64094	86200-52	115127-27	151134-14
	MO	2421	8309	16988-1	28137	42480	61102	83031	112113-1	147322-1
m144	C2.0	7494	18278	30012	45085	63532	88457	121865	163048	217250
	M3.0	6919	17647	29791	43658	63856	89576-1	117613-66	157126-35	205550-16
	MO	6761	16494	27261	40931	60368	84484-1	113908-1	150929-1	201402-1
m14	C2.0	3966	13838	28231	52499	77621	111343	157092	215337	290045
	M3.0	4109	14322	28128	50410	74169	110465-1	152160-1	207093-52	277622-25
	MO	3893	13600	27027	45579	71586	106225	147802	203111	272514-1

Table 2. Experimental results

For comparison purposes, we report tests with version 2.0 of Chaco (C2.0) Hendrickson and Leland (1995) and version 3.0 of Metis (M3.0) Karypis and Kumar (1997), which are both well known graph partitioning software packages. Tests

performed with Chaco use the multi-level algorithm implemented in its software. Tests performed with Metis use the program "pmetis" when the number of sets in the partition is smaller than 65 and the program "kmetis" when the number of sets is larger than 64. We have performed tests with several other partitioners, but their results were generally dominated by these two partitioning packages, therefore for lack of space we don't report the results obtained with the other partitioners.

In Table 2, the first column refers to the graphs tested ("whit" stands for whitaker3 and "btk31" stands for bcsstk31) and the second column refers to the graph partitioning methods used. The other columns refer to the values of the edge cuts according to the number of sets in the partition. A dash followed by a number indicates an imbalance in the sets of the best partition. For example, "-1" indicates an imbalance of one node between the set(s) having the larger number of nodes and the one(s) having the smaller number of nodes. Chaco always reports balanced partition while Metis and our algorithm sometime report a best solution corresponding to an imbalance partition. In a single case (Ocean, 2-way), our results are not better or as good as the two other partitioners. On small graphs and small number of sets, we obtain same solutions or small improvements. For larger graphs or when the number of sets in the partitions increase, we distance ourself from the two other approaches with sometime substantial improvements both in relative and absolute values.

Our algorithm requests a substantial amount of computational time. For the results in Table 2, Chaco and Metis ended their computation 5 to 20 times faster than our algorithm. Generally, our algorithm need about twice more time than Chaco or Metis to get a solution which is better than these partitioners, the rest of the time is spent to improve the quality of the solution. On the other hand, running any other partitioner for as much time as our algorithm (using several different coarsening factors or any other adjustable parameters) didn't get solutions competitive with those shown in Table 2 for our algorithm.

5 Conclusion

The main goal in the design of our current approach was to enhance the control structure of the cooperative procedure (the set of search programs as a whole, their global behavior). Our strategy to achieve this goal has been to simplify the cooperation scheme in terms of the neighborhood structure allowed among cooperating programs (a linear array), to have a more explicit definition of the search space that a process is allowed to explore (the neighborhood functions), and to decrease the probability that shared information will mislead the search programs, by establishing better control over the cost function for selecting reused information (the three inter-task operators). But most importantly, this new design redefines the purpose of the search programs $P_i > 0$ in the multi-level structure. Reused information helps to modify the data sets defining the search space available to the search methods rather than directly influencing the search parameters of the cooperating programs. The main purpose of the search

programs is then to find information useful to neighbor levels in the control structure, rather than to directly generate solutions for the optimization problem. The search programs $P_i > 0$ become part of a complex multi-level control structure which spans several computers. Their goal is to assist the search performed by program P_0, which is the only program that can find explicit solutions to the optimization problem. In this regard, the current design is an example of how we can transform the complex interactions among cooperating programs into a useful explicit control structure of cooperative procedures.

References

1. F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8(1):156–166, 1977.
2. F. Glover. Ejection Chains, Reference Structures and Alternating Path Methods for the Traveling Salesman Problem. Report, University of Colorado, Boulder, 1992.
3. F. Glover. A Template for Scatter Search and Path Relinking. In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, editors, *Lecture Notes in computer Science*, 1997.
4. F. Glover and M. Laguna. Tabu Search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, 1993.
5. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.
6. B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. Report SAND93-1301, Sandia National Laboratories, 1993.
7. B. Hendrickson and R. Leland. The Chaco User's Guide: Version 2.0. Report SAND95-2344, Sandia National Laboratories, 1995.
8. G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, to appear.
9. G. Karypis and V. Kumar. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices: Version 3.0. Report, University of Minnesota, 1997.
10. J.P. Kelly and J. Xu. Tabu Search and Vocabulary Building for Routing Problems. Technical report, Graduate School of Business Administration, University of Colorado at Boulder, 1995.
11. L. Lopez, M.W. Carter, and M. Gendreau. The Hot Strip Mill Production Scheduling Problem: A Tabu Search Approach. Report, Center for Research on Transportation, Université de Montréal, 1996.
12. Y. Rochat and E. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1(1):147–167, 1995.
13. E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A New Neighborhood Structure for Vehicule Routing with Time Window. Report CRT-95-66, Center for Research on Transportation, Université de Montréal, 1995.
14. M. Toulouse, T.G. Crainic, and B. Sansó. An Experimental Study of Systemic Behavior of Cooperative Search Algorithms. In I.H. Osman S. Voss, S. Martello and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 373–392. Kluwer Academic Publishers, 1999.

This article was processed using the LaTeX macro package with LLNCS style