Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, Russia, June 3 - 5, 2009

A Genetic Local Search Algorithm for the Graph Partitioning Problem with Cardinality Constraints ⋆

Yu. Kochetov, A. Mikhailova, A. Plyasunov *

* *Novosibirsk State University, Novosibirsk, Russia (e-mails: jkochet@math.nsc.ru, real_nastya@mail.ru, apljas@math.nsc.ru).*

Abstract:
A new genetic local search algorithm is designed for the graph partitioning problem with cardinality constraint for each subset of the vertices. The family of local optima under the polynomial neighborhoods is used as a population in order to systematically produce better local optima. It is shown that the corresponding local search problems are tightly PLS-complete. So, any local improvement algorithm takes, in the worst case, an exponential number of iterations regardless of the tie–breaking and pivoting rules used. Nevertheless, the local search problems are polynomially solvable if all weights of the edges are identical. For this case, computational experiments are produced for the real–world and random test instances. We observe that this algorithm is efficient and effective. It allows to find the high quality local optima.

Keywords: Graph partitioning problem, local search, memetic algorithm, Min-Cut problem, PLS-completeness.

1. INTRODUCTION

In the well-known graph partitioning problem, we are given an undirected graph $G = (V, E)$ with an even number of vertices and a weight $w_e \geq 0$ for each edge $e \in E$. We wish to find a partition of V into two subsets V_1 and V_2 with $|V_1| = |V_2|$ such that the sum of the weights of the edges that have one endpoint in V_1 and one endpoint in V_2 is minimal. In the real world applications, we need, as a rule, to partition a graph into many subsets under the cardinality constraint for each of them. Moreover, we know nothing about the number of subsets. It is a more complicated model (Chopra, Rao (1993); Rendl, Wolkowicz (2005)). We face with it in our traffic flow project for Vladivostok city, Far East of Russia.

For this NP-hard problem (Gary, Johnson (1979)), we develop a genetic local search algorithm (Dreo et al. (2006)) where elements of population are local optima under the polynomial neighborhoods. We study the corresponding local search problems and show that these problems are tightly PLS-complete. So, any local improvement algorithm takes an exponential number of iterations in the worst case regardless of tie–breaking and pivoting rules used. Nevertheless, these algorithms are fast in average. They are polynomial if all weights of the edges are the same. A lot of test instances for this case are available by internet. We have carried out computational experiments for these random generated test instances and an instance from the map of Vladivostok city. For all cases, we observe that our genetic local search algorithm is efficient and finds high quality local optima.

⋆ This work was partly supported by the RFBR grant 08-07-00037, ADTP grant 2.1.1/3235.

2. THE STATEMENT OF THE PROBLEM

Given an undirected graph $G = (V, E)$ and a weight $w_e \geq 0$ for each edge $e \in E$, find a partition of V into subsets with cardinality of each subset at most p such that the sum of the weights of the edges that have endpoints in different subsets is minimal (Chopra, Rao (1993); Rendl, Wolkowicz (2005)).

Now we present this problem as a linear 0-1 program. Let the set $J = \{1, \ldots, m\}$ indicates the indices of the subsets. Introduce the following 0-1 decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to subset } V_j \\ 0, & \text{otherwise,} \end{cases}$$

$$y_e = \begin{cases} 1 & \text{if edge } e \text{ belongs to the cut} \\ 0, & \text{otherwise.} \end{cases}$$

Using these variables, we can write the problem as follows:

$$\min \sum_{e \in E} w_e y_e$$

s.t.

$$\begin{cases} y_e \geq x_{i_1 j} - x_{i_2 j}, \\ y_e \geq x_{i_2 j} - x_{i_1 j}, \end{cases} \quad e = (i_1, i_2) \in E, j \in J,$$

$$\sum_{j \in J} x_{ij} = 1, \quad i \in I,$$

$$\sum_{i \in V} x_{ij} \leq p, \quad j \in J,$$

$$y_e, x_{ij} \in \{0, 1\}, e \in E, i \in V, j \in J.$$

The objective function defines the weight of the cut. The first set of restrictions requires to include an edge into the cut if its endpoints belong to different subsets. The second

set of restrictions guarantees that each vertex is included into exactly one subset. The last restrictions determine the upper bound for the cardinality of each subset.

This representation allows us to use commercial software, for example, CPLEX, and find optimal solution. Unfortunately, this partitioning problem is very hard for exact methods. We try to solve small random test instances with $|V| = 50$ and probability $0,33$ to have an edge for each pair of vertices. After 12 hours of running time, PC Pentium IV with RAM 512 Mb finds a solution which is strongly worse than a solution obtained by genetic algorithm in a few seconds. The integrality gap is quite large for these instances, about 50 %. So, lower bounds are not tight and the branch and bound strategy is not efficient for this case.

Note that we can rewrite the problem without y_e variables. Clearly, $y_e = 1 - \sum_{j \in J} x_{i_1 j} x_{i_2 j}$, $e = (i_1, i_2)$. Hence, we can remove the first set of restrictions and present the problem as a nonlinear one:

$$\max \sum_{e \in E} w_e \sum_{j \in J} x_{i_1 j} x_{i_2 j}$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \quad i \in I,$$

$$\sum_{i \in V} x_{ij} \leq p, \quad j \in J,$$

$$x_{ij} \in \{0, 1\}, \quad i \in V, j \in J.$$

Now all constraints are linear, but the objective function is quadratic. So, we can use CPLEX software again. We reduce the dimension of the problem, remove some constraints and variables but get bad results again. For the same random test instances, we obtain even worse partitions than for the linear program after 12 hours of running time.

The main reason, we believe, why these programs are not useful, is the existence of lots of symmetries. Arbitrary renumbering of the set J produces a new formal representation of the same solution. In particular, we have at least $|J|!$ optimal solutions. This leads to a huge branching tree and weak lower bounds. Efficient methods for removing the symmetries are unknown. Some results in this direction can be found in (Peinhardt (2008); Bertold (2008)). Now heuristic methods are the most powerful for this partitioning problem, especially for the large scale instances.

3. GENETIC LOCAL SEARCH

The evolution strategies, the evolution programming, and the genetic algorithms are famous approachers in combinatorial optimization. They have went through many modifications according to the variety of problems. The genetic algorithms became extremely popular after the publication of book "Genetic Algorithms in Search, Optimization and Machine Learning", by D.E. Goldberg in 1989. This book, published world–wide, has resulted in an exponential growth of interest to this field. Nowadays, researchers in this area organize common international conferences and combine their different ideas. In this paper, we study genetic local search algorithm (GLS), which is interesting from the theoretical and practical points of view. It is a variant of Memetic algorithm where we apply different greedy strategies and crossover operators. The GLS is an iterative method. At each iteration, we have a set of local optima under the prescribed neighborhoods. This set constitutes a population. It evolves during a succession of iterations until a termination criterion is satisfied. The general framework of this metaheuristic can be presented as follows.

Genetic Local Search

1. Generate a starting population.
2. Repeat the following steps until the termination criterion is satisfied:
  2.1 Select two elements from the population as parents.
  2.2 Create an offspring solution for the parents.
  2.3 Apply local search for the new solution and find local optimum.
  2.4 Update the population.
3. Return the best found solution.

Below we discuss this framework in detail.

3.1 Starting population

If number of iterations of the genetic algorithm is great then the choice of starting population is not crucial for the convergence properties of the algorithm. Nevertheless, the good quality of starting population can reduce the running time to get optimal or target solutions. To this end, we use various greedy strategies and local search at Step 1 of the framework.

Random Greedy. Given $p > 0$, we define the minimal number of subsets for partition, $m = \lceil |V|/p \rceil$, and select one vertex from V for each subset at random. Further, we add vertices to the subsets one-by-one according to the cardinality constraint. At each step, we pick up an unused vertex at random and include it in a subset with the minimal increase of the objective function.

Heavy Greedy. This algorithm is similar to the previous one but uses another rule to select a vertex at each step. Namely, a vertex with the maximal total weight of its edges is selected.

Light Greedy. This algorithm is similar to the Heavy Greedy, but uses an opposite strategy. A vertex with the minimal total weight of its edges is selected.

Unfortunately, these algorithms, as many others, produce solutions with a large relative error. For example, the best known solution for a graph with 2395 vertices, 7462 edges, and $w_e \equiv 1$ has objective value 596 for $p = 1198$. However, Random Greedy produces 3730 on the average, Heavy Greedy produces 1928, and Light Greedy produces 1923. Similar deviations are observed for other graphs and other greedy strategies.

In order to improve greedy solutions, we may use multi-start approach and return the best found solution. Further improvement can be obtained by local search under polynomially searchable neighborhoods. Algorithms based on this idea are called Greedy Randomized Adaptive Search Procedures (GRASP (Festa, Resende (2002))). We apply them to create the starting population of local optima. The performance of the GRASP algorithms strongly depends on the neighborhoods used in the local search. Large

neighborhoods lead to high–quality solutions but spend a lot of time. Small neighborhoods often produce poor local optima. Finding an optimal balance is a non-trivial task. So, we will apply small and large neighborhoods with different greedy strategies.

3.2 Local search

Let us consider the following neighborhoods, which will be used at Steps 1 and 2.3 of the GLS framework.

MS neighborhood (Move or Swap). This neighborhood consists of feasible solutions which can be obtained from the given solution by moving a vertex to another subset (may be empty one) or by choosing two vertices in different subsets and swapping the vertices. Note that we can change the power of subsets of this neighborhood. The size of the neighborhood is $O(|V|^2)$.

KL–neighborhood (Kernighan, Lin (1970)). Elements of this neighborhood are obtained by the following iterative procedure.

Step 1. Find the best element in the MS–neighborhood for the current solution. This element may be better or worse than the current solution.
Step 2. Move from the current solution to the best MS–solution.
Step 3. Repeat Steps 1 and 2 where a vertex cannot be chosen if it has been used in one of the previous iterations of Steps 1 and 2.

The fulfilment of the iterations defines a sequence of feasible solutions. They compose the KL–neighborhood of the current solution. The size of the neighborhood is $O(|V||J|)$.

KL_1–neighborhood. This neighborhood consists of one element only. It is obtained by the first iteration of the previous procedure. By definition, a KL_1–neighborhood is a part of the MS–neighborhood.

FM–neighborhood (Fiduccia, Mattheyses (1982)). This neighborhood is defined like the KL–neighborhood. The only difference is in the rule of the choice of two vertices for swapping. Now this step is divided into two stages. At the first stage, we select the first vertex and a subset of the partition where this vertex will be. At the second stage, we select the second vertex from the subset for swapping with the first vertex. At each stage, we try to minimize the objective function value. Recall that the objective value may decrease or increase at each iteration.

FM_1–neighborhood. It consists of one element only. This element is determined at the first iteration of the procedure which defines FM–neighborhood.

Similar neighborhoods were used for graph bipartition problem (Fiduccia, Mattheyses (1982)), the p–median problem (Kochetov et al. (2005)), the timetabling problem (Kochetov et al. (2008)), and others. From the theoretical point of view, it is interesting to understand the computational complexity of the local search problems for these neighborhoods (Kochetov (2008); Yannakakis (1997)). For $w_e = 1, e \in E$ these problems are polynomially solvable. But it is an open question for arbitrary positive weights. We claim that the local search problem for each of these neighborhoods is tightly PLS–complete for this general case. As a result, we have the following.

Theorem. For the graph partitioning problem with cardinality constraints, the local improvement algorithm under each of the neighborhoods MS, KL, KL_1, FM, FM_1 takes, in the worst case, an exponential number of iterations regardless of the tie–breaking and pivoting rules used.

3.3 Selection and identification

As a rule, the capability of an element of population to be selected for reproduction depends on its objective value. The most popular selection operators are proportional and tournament selections (Dreo et al. (2006)). The main idea of the operators is to give a preference for "good" solutions. In the GLS algorithm, population consists of local optima only. All solutions are not so bad. So, we select parents from the population at random with the uniform distribution.

Let A, B be two elements of the population. We will represent each solution by a string with length $n = |V|$ where the bit i shows the subset for vertex i. As we mentioned above, the representation of solution admits a lot of symmetries. So, we may get identical partitions of set V for different A and B. Moreover, if A and B are different, we can select a numbering of subsets for B that minimizes the difference (distance) between A and B. If the distance equals zero, then A and B coincide.

Let m_{ij} denote the number of vertices which either belong to subset i of A and not belong to subset j of B or *vice versa*. Now we introduce the auxiliary variables

$$z_{ij} = \begin{cases} 1 & \text{if subset } i \text{ is assigned to subset } j \\ 0, & \text{otherwise.} \end{cases}$$

The distance between A and B can be defined as the optimal value of the following assignment problem (Moraglio et al. (2007)):

$$\min \sum_{i \in J} \sum_{j \in J} m_{ij} z_{ij}$$

s.t.

$$\sum_{i \in J} z_{ij} = 1, \quad j \in J,$$

$$\sum_{j \in J} z_{ij} = 1, \quad i \in J,$$

$$z_{ij} \in \{0, 1\}, \quad i, j \in J.$$

This problem is polynomially solvable and produces the best numbering of the subsets for solution B. Below we assume that solution B has been renumbered according to the optimal solution of the assignment problem.

3.4 Generating the local optima

During the evolution process, we need systematically generate new local optima. To this end, a crossover operator is applied to parent solutions (Step 2.2) and the local search algorithm is used (Step 2.3). The crossover operator is usually stochastic. The repeated crossover of the same couple of distinct parents gives different offspring. The operator generally respects the following properties (Dreo et al. (2006)):
1) the crossover of two identical parents will produce

offspring identical to the parents;
2) two parents which are close in the search space will generate offspring close to them.

These properties are satisfied by the "classical" k-point crossover and by the others described below.

Classical k-point crossover. It is a well–known operator, which was applied for many optimization problems (Dreo et al. (2006)). It is simple and intuitive. For solutions A and B, we select k different positions $i_1, \ldots, i_k$ from 1 to n. New solution C will get the first positions $1, \ldots, i_1$ from A, the next positions $i_1+1, \ldots, i_2$ are taken from B, positions $i_2+1, \ldots, i_3$ are taken from A again, and so on. Solution C will inherit some properties of its parents. But this string C may be infeasible. To repair it, we remove some vertices from the overfilled subsets and distribute them among other subsets by greedy algorithms. In our computational experiments, we use this operator for $k = 5$.

Proportional crossover. This operator is quite simple and natural for the problem. If solutions A, B indicate the same subset for vertex i, then solution C saves it. Otherwise, C inherits one of them: subset $j(A)$ from A with probability P_A or subset $j(B)$ from B with probability P_B. These probabilities are defined as follows:

$$P_A = \frac{S_A}{S_A + S_B}, \quad P_B = \frac{S_B}{S_A + S_B},$$

where $S_A(S_B)$ is the total weight of the edges which connect the vertex with its subset in solution $A(B)$. We use this rule according to the cardinality constraint. If one of the subsets $j(A)$ or $j(B)$ is full, the vertex is included to another subset.

Crossover Laszewski. We select some subsets in solution A and save them for C. These subsets will never change, and some properties of A are transmitted to C. Now we remove the vertices of these subsets from solution B and save the other subsets for solution C. In order to get a feasible solution from C, we apply greedy algorithms again. Similar crossover operator is used in (Laszewski (1991)) for the k-way partitioning problem.

Cycle crossover. The crossover of Laszevski tries to keep the following properties of the parents: some vertices of the graph must be at the same subsets. In the cycle crossover, the most attention is attracted to another property: some vertices of graph must be in different subsets.

Let us place string A under the string B and get a permutation with repetitions. We wish to decompose the permutation into cycles and paths. For this goal, we select an arbitrary position in string A and mark it. In this position, we have an index of a subset, say j_0. If the string B has the same index at the same position, then we have got a cycle. Otherwise, the string B has another index, say $j \neq j_0$. Now we select an arbitrary unmarked position of string A with index j and mark it. The process is repeated until either $j = j_0$ or the string A has no unmarked position with index j. In the first case, we have got a cycle; otherwise, we have got a path. Thus, we decompose strings A and B into the cycles and paths. Now to each cycle and path, we assign a label A or B. The offspring C is obtained by picking up an index of subset according to the label for each position. Similar crossover operator is studied in (Moraglio et al. (2007)).

Path relinking. The idea of this operator is suggested in (Glover, Laguna (1997)) and used in many applications. We construct a shortest path from solution A to B and choose an offspring solution C into this path. We create the path by MS-neighborhood. So, each element of the path inherits some properties of the parents. We may get arbitrary internal element as the offspring. In practice, we define it as equidistant element from A and B, i.e. we give equal rights to the parents. Note that there are many shortest pathes from A to B. We use greedy strategy to create one of them.

4. COMPUTATIONAL EXPERIMENTS

The GLS algorithm is coded in PASCAL (Delphi 6.0) and tested both on randomly generated and on the real world instances. In particular, we consider the graph of the traffic roads of Soviet district of Vladivostok city, Far East of Russia. The graph has 217 vertices and 258 edges, $w_e = 1, e \in E$. Our interest to this graph is due to an industrial project on the optimization of traffic flows in this city (Nurminski (2008)). The decomposition approach used in the project requires near optimal solutions for partitioning problem. Table 1 shows computational results for this graph, with various values of parameter p. We test the GLS algorithm with population size 20 and terminate calculations if the population does not change during 20 iterations. The crossover operator is selected at each iteration at random from the list of operators described in the previous section.

Table 1. Results for the graph of roads

p	150	110	100	90	80	60	50	40
Iterations	208	265	450	402	316	293	723	356
GLS	4	5	7	8	10	11	15	16

Table 2. Results for graph with $|V| = 2851$, $|E| = 15093$

p	1426	1497	713	746
Iterations	647	713	1377	889
Best known values	189	181	429	378
GLS	189	181	399	363

After crossover, we apply local improvement algorithm with MS-neighborhood. To the local optimum obtained, we again apply local improvement algorithm with KL-neighborhood. As we can see from the Table 1, the number of cutting edges is quite small. So, the deviation from optimal values cannot be large. Unfortunately, we have not got the optimal values. In order to understand the capabilities of the GLS algorithm, we do as follows. For $p = 50$, we enlarge the size of population from 20 to 50 and terminate the algorithm after 1500 iterations. As a result, we get a better solution, with 13 cutting edges instead of 15. Hence, the GLS algorithm can improve its own partitioning if it gives more possibilities for search. Moreover, the global optimality is not very important in

the decomposition approach: here, we need near optimal solutions quickly.

The tested graph is planar. So, it is interesting to study more complicated cases; for example, random graph from the Graph Partitioning Archive (http://staffweb.cms.gre.ac.uk/∼c.walshaw/partition). For all instances from this archive, the best found solutions are available by internet. Actually, these results are obtained for the case when the number of subsets is given. So, we have to modify our model and introduce one additional restriction for partition. The instances from the archive have large dimension. Hence, the large neighborhoods like KL and Swap are very time–consuming. We use FM and FM_1 neighborhoods only. Moreover, to apply these neighborhoods efficiently, we modify the data structure from (Fiduccia, Mattheyses (1982)) to accelerate the local search. Table 2 presents computational results for a graph with 2851 vertices and 15093 edges.

The case $p = 1426$ is very close to the classical graph bipartitioning problem. We need to divide the set V into two subsets, $V = V_1 \cup V_2$, and the cardinalities of V_1, V_2 differ at most 1 only. In this case, the GLS algorithm finds the best known solution. In the case $p = 1497$, we need to divide V into two subsets again but now there is more freedom. Each feasible solution for $p = 1426$ is feasible for $p = 1497$. So, the optimal value for $p = 1497$ is at most that for $p = 1426$. We observe the same in Table 2. We find the best known solution again and cannot improve it by enlarging the population from 50 to 150 and increasing the number of iterations substantially. We believe that these are the optimal values for $p = 1497$ and $p = 1426$. The cases $p = 713$ and $p = 746$ are more interesting. We have to divide the set V into four subsets, and the assignment problem from Subsection 3.3 is crucial here. For these cases, the GLS-algorithm finds new solutions with better values than the best known solutions. The population size is 50 in all these experiments.

Table 3 shows computational results for a graph with $|V| = 2395$ and $|E| = 7462$. We study five cases: $p = 1256$, 1198, 628, 599, 300. For $p = 1198$ and $p = 300$, we obtain slightly worse results than in the Graph Partitioning Archive. But for the other cases, $p = 1256$, $p = 628$, and $p = 599$, we find better values again. So, we may conclude that our GLS algorithm is efficient and effective for this NP-hard problem and can be used in real–world applications.

Table 3. Results for graph with $|V| = 2395$, $|E| = 7462$

p	1256	1198	599	628	300
Iterations	763	1054	1000	1000	1216
Best known values	551	596	1203	1184	1758
GLS	541	599	1174	1130	1761

5. CONCLUSIONS

We consider the graph partitioning problem with cardinality constraints for the subsets of vertices. For this NP-hard problem, we design an iterative genetic local search algorithm. This algorithm deals with local optima with respect to some polynomial neighborhoods. We show that the general problem of finding a local optimum is tightly PLS-complete. So, any local improvement algorithm takes, in the worst case, an exponential number of iterations regardless of tie–breaking and pivoting rules used. But for the case of identical weights of the edges, we get local optima in polynomial time. For this case, we carry out computational experiments for the real–world and random test instances. We observe that the GLS algorithm is efficient and finds near optimal solutions quickly.

REFERENCES

Bertold T. (2008) Automatic detection of orbitopal symmetries. *Abstracts of International Conference Operations Research 2008, September 3–5, 2008*, Augsburg, Germany, 198–198.

Chopra S., Rao M. R. (1993) The partition problem *Mathematical Programming*, 59, 87–115.

Dreo J., Petrowski A., Siarry P., Taillard E. (2006) Metaheuristics for Hard Optimization, Springer.

Festa P., Resende M. G. C. (2002) GRASP: An annotated bibliography. In: C. C. Ribeiro, P. Hansen (Eds.) *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, 325–368.

Fiduccia C. M., Mattheyses R. M. (1982) A linear-time heuristic for improving network partitions *Proc. of the 19-th Design Automation Conference, Los Alamitos, CA* IEEE Comput. Soc. Press, 175–181.

Gary M. R., Johnson D. S. (1979) Computers and Intractability: A Guide to the Theory of NP–Completeness. New York: Freeman.

Glover F., Laguna M. (1997) Tabu Search. Boston: Kluwer Acad. Publ.

Kernighan B. W., Lin S. (1970) An effective heuristic procedure for partitioning graphs *Bell System Techn J.*, 49, 291–307.

Kochetov Yu., Alekseeva E., Levanova T., Loresh M.(2005) Large neighborhood local search for the p-median problem *Yugoslav Journal of Oper. Res.*, 15(1), 53–63.

Kochetov Yu., Kononova P., Paschenko M. (2008) Formulation space search approach for the teacher/class timetabling problem *Yugoslav Journal of Oper. Res.*, 18(1), 1–11.

Kochetov Yu. (2008) Computational capabilities of local search in combinatorial optimization. *Computational Mathematics and Mathematical Physics*, 48(5), 747–763.

Laszewski G. (1991) Intelligent structural operators for the k-way graph partitioning problem. *Proceedings of Fourth International Conference on Genetic Algorithms, San Diego, CA, USA*, 45–52.

Moraglio A., Kim Y.-H., Yoon Y., Moon B.-R. (2007) Geometric crossovers for multiway graph partitioning, *Evolutionary Computation*, 15(4), 445–474.

Nurminski E.A. (2008) Decomposition and parallel computations via Fejer processes with small disturbances. *Proceedings of XIV Baikal International School–Seminar Optimization Methods and Their Applications*, 1, 128–135.

Peinhardt M. (2008) Breaking model symmetry for graph partitioning. *Abstracts of International Conference Operations Research 2008, September 3–5, 2008, Augsburg, Germany*, 198–198.

Rendl F., Wolkowicz H. (2005) A projection technique for partitioning the nodes of a graph. *Annals of Operations Research*, 58(3),155–179.

Yannakakis M. (1997) Computational complexity. In: E. Aarts, J. K. Lenstra (Eds.) *Local Search in Combinatorial Optimization*, Chichester: Wiley, 19–55.