International Congress on Evolutionary Methods for Design,
Optimization and Control with Applications to Industrial Problems
EUROGEN 2003
G. Bugeda, J.A.- Désidéri, J. Periaux, M. Schoenauer and G. Winter (Eds)
©CIMNE, Barcelona, 2003

MOVE-BASED CROSSOVER FOR GRAPH PARTITIONING PROBLEMS

Andreas Rummler

Technical University of Ilmenau
Department Electronic Circuits and Systems
PO Box 100565, 98684 Ilmenau, Germany
e-mail: rummler@evolvica.org, web page: http://www.inf-technik.tu-ilmenau.de

Key words: graph partitioning, multiobjective optimization, crossover, SPEA

Abstract. *This paper introduces a specialized crossover operator for use in graph partitioning problems. The article starts with a short introduction to the graph partitioning problem and its formulation as a multi-objective task. Thereafter a suitable genetic representation, that is capable of ignoring redundant solutions from the search space is introduced. This encoding works together with the proposed crossover operator, that borrows its main idea from conventional move-based partitioning heuristics. The operator has been incorporated into SPEA and has been tested with several benchmark graphs. Experimental results are shown in the last section of the paper.*

1 Introduction

Partitioning is a significant problem in various engineering disciplines. It plays a key role in designing computer systems in general and VLSI chips in particular. In most cases graphs (or as generalization hypergraphs) are used for modelling electronic circuits and systems. Therefore, the partitioning of a whole system can be traced back to a graph partitioning problem.

The problem of partitioning a hypergraph can be formulated as follows: Let $H = (V, E)$ be a hypergraph with a non-empty set of vertices V and a set of hyperedges E. Each vertex $v_i \in V$ is assigned to one of the subsets (partitions) $V_1 \dots V_k$ such that $V_i \bigcap V_j = \oslash$ for $i \neq j$. As a consequence a set of hyperedges $E_C \in E$ exists that contains all hyperedges that have at least two incident vertices that belong to different vertex subsets. This set of hyperedges is refered to as the *cutset*. The *cutsize* $c(H) = |E_C|$ equals the number of hyperedges belonging to the cutset. The aim of partitioning the hypergraph is to find a partition assignment of all vertices such that the cutsize becomes minimal.

The graph partitioning problem can be seen as a multi-objective optimization problem. The most important criterion is of course the minimization of the cutsize $c(H)$. In other words the purpose is to minimize the number of hyperedges $|E_C|$ that have incident vertices belonging to different partitions. The incidence of a hyperedge e_i with a vertex v_j is denoted by $e_i \leftrightarrow v_j$. This criterion can be written as

$$\begin{aligned} c(H) = |e_i \in E| \quad &\Rightarrow \text{minimal} \\ \text{with} \quad \exists\, e_i \leftrightarrow v_j, e_i \leftrightarrow v_k \quad \text{and} \quad v_j \in V_m, v_k \in V_n \quad &\text{and} \quad V_m \cap V_n = \emptyset \end{aligned} \tag{1}$$

An algorithm trying to minimize only the cutsize would produce one particular solution: all vertices assigned to the same partition. This is a very good solution in terms of cutsize (which is zero), but of course a useless one. That is why most algorithms include a heuristic to preserve or restore the balance of the solution. Balancing a solution means that all partitions have nearly the same size resp. contain nearly the same number of vertices. It is important to mention that the partitions *should* have same sizes but need not. Unlike in conventional approaches where the balance is often treated as a constraint it is possible to turn the balance into a second criterion. This criterion can be formulated as

$$max\left(\left|\frac{\frac{|V|}{k} - |V_i|}{\frac{|V|}{k}}\right|\right) \quad \forall \quad 0 < i < k \tag{2}$$

with k being the number partitions the graph is divided into.

Additional criteria may be formulated based on the partitioning task originating from the concrete real-world problem. Especially in VLSI design other criteria are conceivable, like for instance the minimization of the signal delay between different partitions.

2 Genetic Representation

For the graph partitioning problem a vector-like genetic representation seems to be suitable. Each vertex can be assigned an integer number indicating the index of the partition the vertex belongs to. Therefore a suitable genetic representation is a vector of

integer numbers with the length equal to the number of vertices in the hypergraph. This scheme is shown in figure 1 (hyperedges with 2 neighbours are marked with straight lines while hyperedges with more than 2 neighbours are marked with a black dot and straight lines).

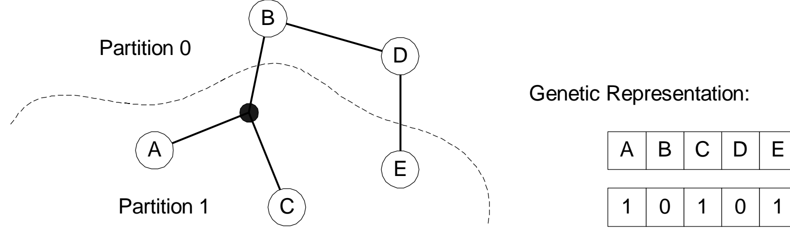

Figure 1: A hypergraph divided into two partitions with its appropriate genetic representation

Unfortunately this kind of representation has got a disadvantage which is not obvious at first sight – the representation contains redundant information that may slow down the search process and prevents a specialized recombination operator from identifying good or bad attributes in parent chromosomes. In addition to the configuration from figure 1 $\{1, 0, 1, 0, 1\}$ there is another configuration that is equal in its information content: $\{0, 1, 0, 1, 0\}$. In the case that the number of partitions is 2, 50% of all possible configurations are redundant. This becomes even worse with an increasing number of partitions (for 3 partitions the following configurations reflect the same case: $\{0, 1, 2, 0, 1\}$, $\{1, 2, 0, 1, 2\}$, $\{0, 2, 1, 0, 2\}$ etc.). This is because the partition numbers can be moved cyclically throughout the partitions.

With this knowledge the entire search space can be divided in two regions: a region containing only feasible solutions and a (substantially larger) region containing redundant solutions. Each solution in the redundant region can be non-ambiguously mapped to exactly one solution in the feasible region. This remapping can be performed with the algorithm shown in algorithm 1.

Algorithm 1 remapping of solutions to their non-redundant solution in the feasible region

define C as array of integers being the chromosome vector
define R as array of integers containing the remapping ($R[i]$ is the new partition number for partition i)
define $index = 0$ as the new number that is assigned to a partition
clear R
for all index $i = 0$ to $length(\mathrm{C}) - 1$ **do**
  if remapping for $C[i]$ does not exist **then**
    define $index =$ new remapping of $C[i]$ and store it in $R[C[i]]$
    increment $index$
  end if
  assign $C[i]$ the remapped number of $C[i]$ (that is $C[i] = R[C[i]]$)
end for

To ensure the validity of a solution at any time this remapping must be performed each time a genetic operator alters the genetic representation. Of course this costs some

computing time, but because of the simplicity and linear complexity of the remapping algorithm this time can be neglected against the crossover and mutation.

3 Move-Based Crossover

To be able to create a well-performing evolutionary algorithm it is essential to develop operators that are tailored to the optimization problem that is to be solved. Simple multi-point crossover of binary strings, that is still used in various cases, often does not perform well in real world applications.

In earlier work ([RA02]) the author has shown that evolutionary approaches can be combined with existing partitioning heuristics. Such heuristics can be transformed into local improvement operators to support the evolutionary search process.

Common graph partitioning heuristics incorporate move-based approaches, such as [San89]. It is obvious to incorporate such a move-based approach into a specialized evolutionary operator. The main idea of the operator introduced here is the following: if a vertex in chromosome configuration A belongs currently to partition i, does a move of this vertex to partition j from chromosome configuration B improve anything?

An algorithm for a crossover operator based on this idea is given in algorithm 2.

Algorithm 2 move-based crossover operator

define A and B as the parent chromosome vectors containing current partition numbers
assign the partition numbers from A to all vertices
for all index $i = 0$ to $length(\mathrm{A}) - 1$ **do**
  calculate gain in cutsize γ_i for vertex i being moved to partition $B[i]$
  calculate gain in incidence degree δ_i for vertex i being moved to partition $B[i]$
  if $\gamma_i < 0$ **then**
    move vertex i to partition $B[i]$ (immediate improvement)
  else if $\gamma_i = 0$ and $\delta_i < 0$ **then**
    move vertex i to partition $B[i]$ (improvement MIGHT be possible later)
  else
    do nothing (no improvement)
  end if
end for

The crossover operator works as follows: first the partition numbers from the first parent chromosome are assigned to the vertices in the hypergraph ($A[i]$ contains the partition number for vertex i). Now for each vertex a check is performed if moving this vertex to partition $B[i]$ improves anything. For being able to make this decision two values are calculated: the gain in cutsize γ and in incidence degree δ. The value of γ is defined as the improvement in cutsize that is immediately achieved when moving a vertex. The gain may be negative or positive. The value of δ is defined as the improvement in the number of *different* partitions of all adjacent vertices of a vertex i. This value must be lowered as well, because a vertex can only be removed from the cutset if all adjacent vertices are contained in the same partition (the incidence degree is 2 in this case).

The two key operations in algorithm 2 are shown below in algorithms 3 and 4 in detail. Algorithm 3 shows the calculation of the gain in cutsize γ. The algorithm iterates over all incident hyperedges of a vertex v. For each incident hyperedge e the number of different

Algorithm 3 cutsize gain calculation

```
define v as vertex to be moved
define s as partition v is currently assigned to (source partition)
define t as partition v may be moved to (target partition)
define n_d(e) as number of different partitions of all incident vertices of edge e
γ = 0
if s != t then
  for all incident edges e of v do
    calculate n_d(e)
    if n_d(e) = 1 then
      increment γ
    else if n_d(e) = 2 then
      if v = only vertex in different partition than all other incident vertices of e then
        decrement γ
      end if
    else
      do nothing: no improvement possible
    end if
  end for
end if
```

partitions of all incident vertices $n_d(e)$ is calculated. If this value is equal to 1 the gain would be increased because moving v would introduce e to the cutset. Only if $n_d(e)$ is equal to 2 and v is in a different partition than its adjacent vertices in e the gain can be improved (decremented). If $n_d(e)$ is bigger than 2 no improvement is possible.

Algorithm 4 incidence degree gain calculation

```
define v as vertex to be moved
define s as partition v is currently assigned to (source partition)
define t as partition v may be moved to (target partition)
define n_t(e) as number of different partitions of all incident vertices of edge e
δ = 0
if s != t then
  for all incident edges e of v do
    calculate n_s(e) as number of incident vertices of e that are in the same partition as v
    calculate n_t(e) as number of incident vertices of e that are in the same partition as v
    if n_s(e) = 1 then
      decrement δ
    end if
    if n_t(e) = 0 then
      increment δ
    end if
  end for
end if
```

Algorithm 4 shows the calculation of the gain in incidence degree. It works in a similiar manner as the calculation of the gain in cutsize. Again, the algorithm iterates over all incident hyperedges of a vertex v. For each hyperedge the number of adjacent vertices of v that are in the same partition ($n_s(e)$) resp. that are in the partition v may be moved into ($n_t(e)$) are calculated. Now there are two cases where the gain changes: the first

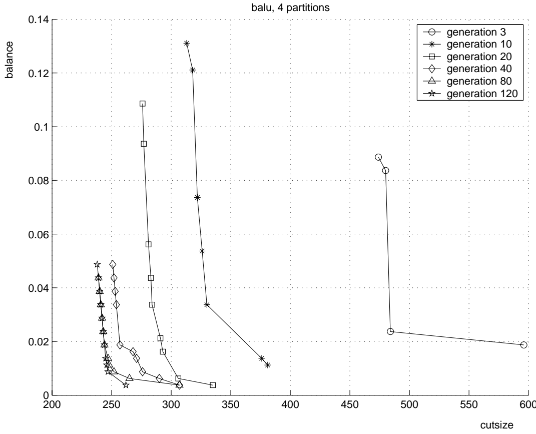

Figure 2: pareto front progress, balu, 4 partitions

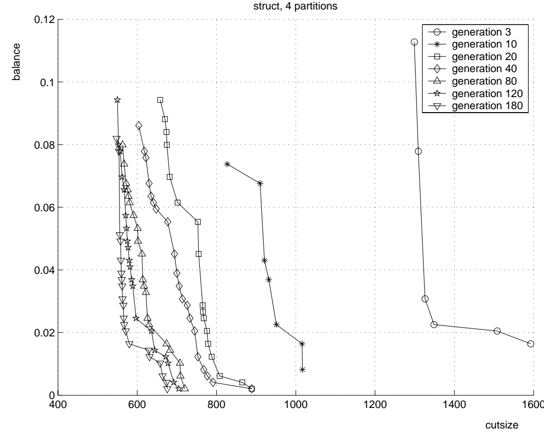

Figure 3: pareto front progress, struct, 4 partitions

one is the case where $n_s(e)$ is equal to 1. The vertex v is the only vertex of all adjacent vertices of e in its partition, so moving away this vertex improves (decrements) the gain. The second case is the case of $n_t(e)$ being 0. This denotes that v will be moved into a partition that has no incident vertices in e and will therefore deteriorate (increase) the gain.

4 Implementation & Results

For testing purposes the proposed crossover operator has been used inside SPEA [Zit99]. The whole algorithm has been implemented using the eaLib toolkit available from [EAL]. As data input hypergraphs from the benchmark suite available from [Cir] have been used.

The algorithm parameters were set as follows:

- population size: 100
- archive size: 20
- selection: tournament
- mutation: scrambling of 50% of chromosome vector, with propability of 15%
- termination: average cutsize of individuals in archive is measured over the last 20 generations, if the change of this average value drops under 0.1, the algorithm is terminated

Figures 2 – 5 show the progress of the pareto front of some of the benchmark graphs. The figures show the cutsize on the x-axis and the balance on the y-axis. Both values must be minimized – the pareto front contracts towards the lower left corner of the diagram. The characteristics of the graphs are as follows:

- *balu* – 735 vertices, 801 hyperedges, 2697 pins[1]

[1] in this context a 'pin' is a 'connection' between a vertex and a hyperedge, so the overall number of pins is equal to $\sum_{i=1}^{|E|} |e_i|$

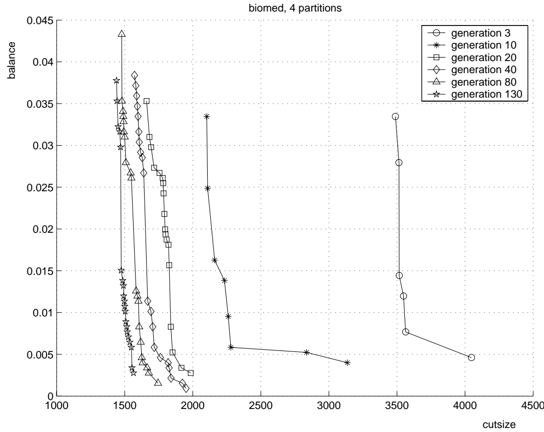

Figure 4: pareto front progress, biomed, 4 partitions

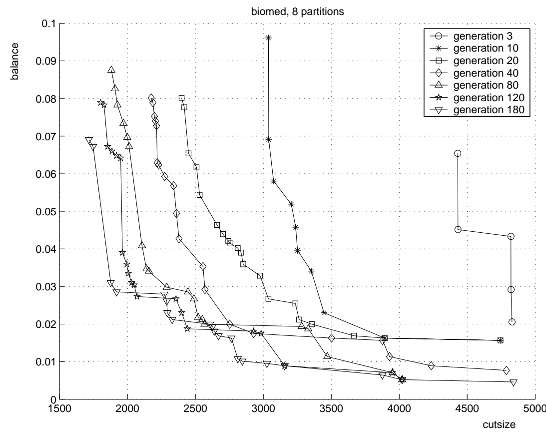

Figure 5: pareto front progress, biomed, 8 partitions

- *struct* – 1920 vertices, 1952 hyperedges, 5471 pins
- *biomed* – 6514 vertices, 5742 hyperedges, 21040 pins

The algorithm converges after approx. 80-120 generations, more or less independent of the size of the problem. This indicates that the introduced recombination operator manages to push the pareto front forward until a certain point, but fails to progress further. One may expect that convergence occures later in bigger problems, but this is not the case and is beyond doubt a problem of the recombination operator.

After convergence of the algorithm a final solution can be selected from the pareto set. An obvious mechanism may be the selection of the solution with the biggest partition being not larger than 5% more than the optimal size of a partition (resp. the smallest solution not being smaller than 5% less). Experiments show that this multi-objective approach can compete with genetic approaches such the one introduced in [MR99], but fails to compete with certain non-evolutionary approaches like [Kar99], which is highly specialized in repetitive bipartitioning, where the minimal cutsize is lower by factor 3-4.

5 Conclusion

The paper describes a move-based recombination operator for graph-partitioning problems. The operator is described in detail and has been implemented using the eaLib toolkit. Experiments with several benchmark graphs show the principle functionality but the performance can't compete with specialized partitioning heuristics. For this reason the introduced crossover operator may not be used in simple partitioning problems, where the minimizing cutsize is the main goal. But it may serve as the basis for recombination operators for multiobjective graph partitioning problems, where the cutsize is only one of several criteria with similiar importance. Improving the overall performance of the operator is subject to future work.

REFERENCES

[Cir] The circuit partitioning page. http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html. URL time: January 20th, 2002, 10^{00}.

[EAL] ealib. http://www.inf-technik.tu-ilmenau.de/~rummler/eng/ealib.html. URL time: August 28th, 2001, 14^{30}.

[Kar99] George Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report 99-034, University of Minnesota, Department of Computer Science, 1999.

[MR99] Pinaki Mazumder and Elizabeth M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation.* Prentice Hall PTR, 1999.

[RA02] Andreas Rummler and Adriana Apetrei. Graph partitioning revised - a multi-objective perspective. In *Proceedings of the 6th Conference on Systemics, Cybernetics and Informatics*, volume 5, pages 163–168. International Institute of Informatics and Systemics, Orlando, USA, 2002.

[San89] Laura A. Sanchis. Multi-way network partitioning. *IEEE Transactions on Computers*, 38(1):1384–1397, 1989.

[Zit99] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD thesis, Eidgenössische Technische Hochschule Zürich, 1999.