A Parallel Implementation of Evolutionary Divide and Conquer for the TSP.

#C L Valenzuela, A.J Jones

#University of Teeside, UK Imperial College London, UK

Abstract.

Evolutionary divide and conquer (*EDAC*) applied to the geometric travelling salesman problem uses a genetic algorithm to explore the space of problem subdivisions. The underlying techniques for subdivision and patching are based on the cellular dissection algorithms of Richard Karp, but the addition of new repair heuristics as well as the genetic algorithm, appears to have succeeded in lifting the quality of solution way above Karp's algorithms, whilst maintaining almost linear scaling of the algorithm with increased problem size. In this paper we outline a parallel implementation of *EDAC* and present some preliminary results of this work.

Introduction.

Experiments with genetic algorithms using permutation operators applied to the Travelling Salesman Problem (TSP) tend to suggest that these algorithms fail in two respects when applied to very large problems: they scale very poorly as the number of cities n increases, and with all except the most sophisticated implementations, the solution quality degrades rapidly (see [Valenzuela 1995a] for an account). In an earlier paper [Valenzuela 1994] we suggested an alternative approach for genetic algorithms applied to hard combinatoric search which we called *Evolutionary Divide and Conquer* (*EDAC*). This technique has potential for any search problem in which knowledge of good solutions for subproblems can be exploited to improve the solution of the problem itself. The idea is to use the genetic algorithm to explore the space of *problem subdivisions* rather than the space of solutions themselves.

The first step in realising an approach along these lines is to examine heuristic TSP algorithms based on the obvious intuitive principal of breaking the problem into subproblems, solving the subproblems and patching the subsolutions (subtours) together to provide a global solution to the original problem. Although several approaches were considered, the cellular dissection algorithms of Richard Karp [Karp 1977] seemed to hold the most promise and were therefore chosen as a starting point. Not only do these algorithms possess an attractively simple geometrical approach to dissection, but they also offer reasonable guarantees of performance. The main challenge is to incorporate these heuristics into a successful *reproductive plan*.

Two particularly novel ideas are incorporated into the *EDAC* approach, the first and most obvious of which being the application of genetic algorithms to the exploration of the space of TSP *problem subdivisions* already discussed.

The other principal novel idea is an alternative approach to the construction of Karp-like tours, viewing them as being recursive, rather than patching globally, as Karp does. This new approach has allowed the incorporation of additional heuristics at a low cost in terms of execution time. Patching in *EDAC* is done whilst threading back up through the recursive subdivisions, and each application of recursive patching is followed by a procedure which we have called *repair*. *Repair* is essentially a combinatoric local tour improvement routine. When n becomes large there is a strong need to reduce any global procedures and the idea that patching should be done recursively yields significant dividends in limiting the combinatoric explosion of cases to be considered for repair.

A particular benefit of *EDAC* is that the model is intrinsically parallel. The overall design lends itself to parallelisation at several levels and in a number of different ways depending upon the parallel architecture. A parallel version of *EDAC* running on the AP1000 supercomputer is presented in this paper.

The Cellular Dissection Algorithm.

Subdivision.

Let rectangle R contain m cities. Let y be the y-coordinate of the centre of R. A horizontal cut through y subdivides R into two equal rectangles, an upper rectangle and a lower rectangle. With uniform random points the effect is to place about half the cities either side of the bisecting line. In a similar fashion, a vertical cut could be applied to bisect the cities through x,

Genetic Algorithms in Engineering Systems: Innovations and Applications
12-14 September 1995, Conference Publication No. 414, © IEE, 1995

which is the x-coordinate of the centre of R.

In cellular dissection algorithms based on Karp's approach, the bisection process is called repeatedly in a recursive fashion in order to subdivide all the new rectangles as they are formed until the number of cities within a particular rectangle falls below a certain threshold, when the subproblems in each rectangle are solved.

In *EDAC* the above bisection technique is modified slightly to make the process of patching the solutions to the subproblems easier.

- Rectangles are bisected through the city nearest to the true area bisection line.

In Karp's algorithms the direction of the cut is always parallel to the shorter side of the rectangle. Karp showed that by minimizing the lengths of the perimeters of the rectangles he was able to minimise the expected lengths of the tours. In *EDAC* the direction of cut is determined at each stage by the genetic algorithm.

Solving the subproblems.

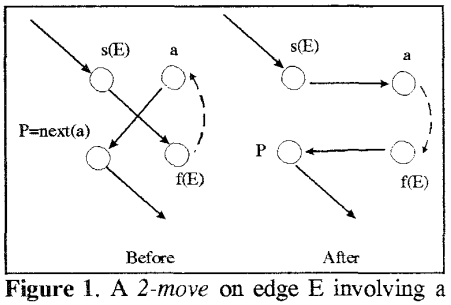

Figure 1. A *2-move* on edge E involving a neighbour *a*.

The *Lin k-opt* heuristics [Lin 1965] are used for solving the subproblems, *Lin 2-opt* being used in the earliest versions of *EDAC* and *Lin 3-opt* for some of the later versions. In our experiments the average subproblem size was only about 8 cities, and the method of solution proved not to be critical.

The *k-opt* algorithms are examples of local tour improvement procedures. They operate on some current tour by exchanging edges that are in the current tour for better edges that are not in the current tour. In the case of *2-opt*, for example, edges are considered for exchange in pairs. In *3-opt* edges are considered in triples. When any such exchange produces a shorter tour length it is made permanent. A single exchange is called a *k-move*. A successful *2-move* is illustrated in Figure 1. For any fixed value of k, the corresponding *k-opt* algorithm always keep on running until a situation is reached where no further improvements are possible by a single *k-move*.

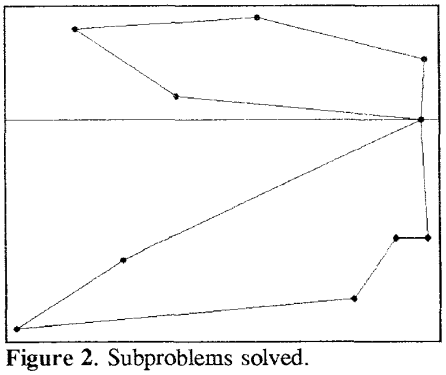

Figure 2. Subproblems solved.

Patching the subtours together.

After the subproblems have been solved, as in Figure 2, the four incident edges to the shared city must be reduced to two. This is achieved by the removal of two of the incident edges, one from each subproblem, and the creation of a new edge between the two "stranded" cities. As there are only four possible ways this patching can be done, they are all tried and one that results in the shortest patched tour is selected. For later purposes the new edge can be added to an edge list L as a candidate for repair.

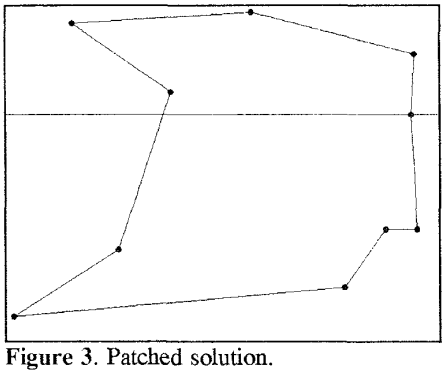

Figure 3. Patched solution.

Recursive fast-repair.

The *recursive fast-repair* heuristics developed for *EDAC* are based on fast versions of the *k-opt* algorithms briefly described above. Various shortcuts have been employed in an attempt to speed things up, several of which were first suggested by [Martin 1992], the most important

```
Procedure Fast-2-repair(T, L, Neighbourhood lists)
   {T is the current tour, L = L(T) is list of edges of T to be considered. Max is the
   maximum edge length of the current tour, Min is the estimated minimum edge length
   of all edges. l is the length of the neighbour lists (l = 10 in these experiments). s(E) is
   start city of edge E, f(E) is final city of edge E. next(a) and prev(a), for city a, are next
   city and previous city, respectively of current tour T}
begin

while L ≠ Ø do
   select edge E = (s(E), f(E)) ∈ L
   m := 1: improvement := false
      while m ≤ l and (improvement = false) do {check neighbours of s(E) & f(E)}
         a := neigh(s(E),m):              {mth neighbour of s(E) on list}
         b := neigh(f(E),m)               {mth neighbour of f(E) on list}
         if (d(s(E),a) + Min > d(s(E),f(E)) + Max) and
            (d(f(E),b) + Min > d(s(E),f(E)) + Max) then break inner while loop
         {check neighbour of s(E)}
         if d(s(E),a) + d(f(E),next(a)) < d(s(E),f(E)) + d(a,next(a)) then
            L := L - {E,(a,next(a))} + {(s(E),a),(f(E),next(a))}
            make 2-move on T              {see Figure 1}
            update Max
            improvement := true
         {check neighbour of f(E)}
         if d(s(E),prev(b)) + d(f(E),b) < d(s(E),f(E)) + d(prev(b),b) then
            L := L - {E,(prev(b),b)} + {(s(E),prev(b)),(f(E),b)}
            make 2-move on T
            update Max
            improvement := true
         m := m + 1                       {no 2-moves, check next neighbour}
      end while                           {take next edge in L}
   L = L - {E}                            {delete edge from the active list}
end while

end
```

Algorithm 1. *Fast 2-repair.*

probably being the restriction of cases of *k-moves* to be considered. For the geometric TSP, when using *k-opt* it is silly to consider sets of edges which are far apart in the physical space of the problem. One way in which *fast-k-opt* makes this idea precise is by maintaining, for each city, a list of the 10 nearest neighbours (say), and restricting *k-moves* to these edges. Algorithm 1 outlines *fast-2-repair*. We refer the reader to [Jones 1995] and [Valenzuela 1995a] for implementation details of *fast-3-repair*.

Recursive fast-k-repair succeeds each simple patching operation in the recursive construction of the global tour, expending most of its efforts repairing small subproblems, where accepted *k-moves* require only short subtour manipulations. In addition each call to *recursive fast-k-repair* is initiated with an edge list L containing just one edge, the rogue edge produced by a single simple patching operation.

Far-repair.

With a view to further improving the solution quality a low-cost tour improvement heuristic has been employed. The ideas used here are similar to those embedded in the *Or-opt* algorithm [Or 1976]. In essence the scheme deletes single cities, or small groups of cities, from their positions in the current tour and inserts them in new positions whenever this move produces a reduction in the tour length. The algorithm, which we call *far-repair*, is applied globally following the construction of

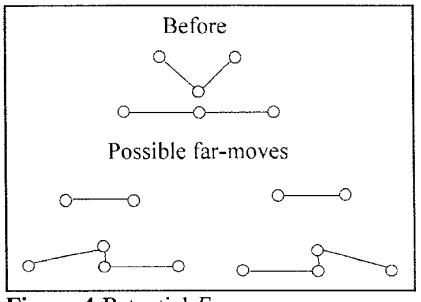

Figure 4 Potential *Far-moves*.

the initial tour by simple patching and *recursive fast-k-repair*.

The lists of nearest neighbours accumulated for the *k-move* procedures are employed by *far-repair* to ensure that the algorithm does not waste valuable time evaluating potential moves that have little chance of success. Figure 4 shows how a single city is tested as a candidate for a *far-move*. It is tried first one side of a near neighbour, then the other. The term 'far' repair refers to the fact that individual cities can be moved to new positions in the current tour that are "far away" from their present positions in terms of where they are on the permutation list defining the tour. *Far-repair* involves exchanging three edges (a *3-move*) and, because it is applied globally, it will repair defects which are beyond the scope of *recursive fast-k-repair*. Further details of *far-repair*, which obviously has time complexity O(n), can be found in [Valenzuela 1994].

The Parallel Genetic Algorithm.

A parallel supercomputer, the Fujitsu AP1000, was chosen as an implementation environment on which to conduct experiments on very large problems. This is an MIMD configuration with one processor, the *host*, coordinating the activities of up to 1024 other processors, the *cells*. The machine we use in this study has only 128 cells.

Grefenstette's master-slave synchronous model [Grefenstette 1981] was chosen as a framework for the parallel implementation of *EDAC*. In the algorithm each of the 128 cells is responsible for evaluating a tour for an "individual" member of the current population. This entails the application of a complete divide-and-conquer procedure on the TSP problem given to the *EDAC* algorithm. The host computer is responsible for computing every other aspect of the genetic algorithm, including selection, reproduction and implementation of the genetic operators crossover and mutation. The master-slave model is appropriate because most of the time during execution of *EDAC* is taken up with evaluating the divide-and-conquer procedure on the various members of the population, and very little time is spent on all other aspects of the genetic algorithm. For example when applying *EDAC* (with *3-repair*) to an 100 city problem Sun serial computers spend more than 99.9% of their execution time on divide-and-conquer evaluations and less than 0.01% on all other aspects of the genetic algorithm. When *EDAC* is applied to a 1000 city problem the proportion of time spent on divide-and-conquer evaluations rises to 99.99%.

The fact that the AP1000 implementation of *EDAC* is controlled by a sequential genetic algorithm meant that the GA previously used on the serial platform required very little modification. The only difficulty encountered with its implementation was due to the "steady state" nature of the population inherent in the weaker parent replacement scheme used previously. In this scheme, based on an algorithm by [Cavicchio 1970], offspring are generated one at a time and a decision made either to replace one or other parent or let the new individual die before the next parental pair are selected for mating. On the AP1000 although the offspring are generated *sequentially* on the host, they are evaluated in *parallel* by the cells.

Each parent and offspring consists of a two dimensional lookup table which when superimposed on the TSP represents decisions to make either vertical or horizontal cuts; the component selected from the lookup table in each case being the one most closely corresponding to the geometric centre of the candidate rectangle. The crossover and mutation of these two dimensional binary arrays are fairly straightforward to implement. (For more details of the genotype and genetic operators see [Valenzuela 1994]).

The main task executed by the cells is, on receipt of a genotype from the host, for each to carry out a divide-and-conquer procedure and return a tour length to the host as a measure of fitness. Thus following the execution of the divide-and-conquer routine by all the cells, the host temporarily possesses *two* populations, one of parental genomes and the other of their offspring. It is not possible to implement the conventional weaker parent replacement model to merge these two groups of individuals,

```
Procedure Parallel GA
begin
   Generate N random structures {N is the population size}
   host sends N structures, one to each of N cells
   cells evaluate tour length produced by each structure
   host collects N tours and N tour lengths, one form each cell
   host evaluates and stores best-so-far
   for generation = 1 to no_generations
      repeat for each population structure
         select next (first) structure
         select a second structure stochastically from a ranked distribution
         apply crossover to produce offspring
         apply mutation to offspring
      until no more population structures
      host sends N structures, one to each of N cells
      cells evaluate tour length produced by each offspring
      host collects N tours and N tour lengths, one form each cell
      repeat for each offspring structure
         if offspring better than occupant of weaker parent slot
            then it replaces it in population
         if offspring better than best-so-far then it replaces best-so-far
      until no more offspring structures
   endfor {generation loop}
   print best-so-far
end.
```

Algorithm 2

because a potential conflict arises each time individual parents produce multiple offspring. Assuming the new offspring are processed sequentially, once a particular parent has been replaced by a stronger first offspring, the parent is not then available for replacement should the algorithm encounter a second offspring from that parent.

Fortunately, the matter is easily resolved. Stronger offspring are simply allowed to occupy the fitness slots vacated by their weaker parents. Thus subsequent offspring are assessed according to the parental fitness slots occupied at the time and the strong elitist strategy inherent in the earlier serial GA is maintained (see Algorithm 2).

Some Results.

Quality of solution.

Early studies demonstrated that all the *EDAC* algorithms reliably produced much better solutions than comparable random search on a trial for trial basis. These control trials were set up using divide, conquer and repair heuristics that were identical to the corresponding *EDAC* algorithm, the only difference being that the direction of bisection for each rectangle that arose was selected at random instead of reading it from a genome lookup table. In fact these experiments not only established that the genetic algorithm had an important role to play, but they also demonstrated that the significance of this role increased with increasing problem size. For example on a 500 city problems an *EDAC* algorithm with *3-repair* (population of 128 for 100 generations) yielded a solution only 0.63 standard deviations better than the best obtained in the comparable random search, whilst for 2000 cities the same *EDAC* algorithm (population of 128 for 300 generations) produced a solution 2.81 standard deviations better than random search. The fact that the results yielded by the genetic algorithm were superior to those produced by the original Karp algorithms was also established early on.

Table 1 summarises the results obtained so far for parallel *EDAC* on uniform random points each with expected solutions of 100 "Stein units" explained in [Valenzuela 1994].

The values entered in the *EDAC* columns each

Problem size	Held-Karp lower Bound	*EDACII* (*2-repair*)	*EDACIII* (*3-repair*)
500	96.01	100.55	100.62
1000	95.22	99.74	99.64
2000	94.30	99.20	98.71
5000	93.59	99.52	99.07

represent the mean best tour lengths obtained from three runs, except in the case of 5000 cities where values are recorded for single runs.

The *EDAC* results are compared with a problem specific measure called the Held-Karp lower bound [Held 1970], [Held 1971], see [Valenzuela 1995b] for an account of this measure. *EDACII* and *EDACIII* represent versions with *2-repair* and *3-repair* respectively.

Time complexity and run-time.

For problems of in the range 500-5000 cities, the scaling exponents, e from $O(n^e)$, for both *EDACII* and *EDACIII*, measured by timing one generation of the GA for each on the AP1000, was almost linear, being 1.04 for *EDACII* and 1.11 for *EDACIII*.

Run-times on the AP1000 for 300 generations of *EDACII* and *EDACIII* on the 5000 city problem were approximately 12 and 28 hours respectively.

Conclusion.

It is our opinion that *EDAC* algorithms show a great deal of potential for solving large travelling salesman problems on parallel hardware. Although the solution quality typically obtained by *EDAC* so far appears to fall slightly short of values obtained by more standard heuristic techniques such as iterated Lin-Kernighan [Johnson 1990], the near linear scaling exponent yielded by *EDAC* on the range of problem sizes studied vindicates further investigations.

References.

[Cavicchio 1970] D.J. Cavicchio. *Adaptive search using simulated evolution.* Unpublished doctorial dissertation, University of Michigan, Ann Arbor 1970.

[Grefenstette 1981] J.J. Grefenstette. *Parallel adaptive algorithms for function optimization.* (Technical Report No. CS-81-19). Nashville: Vanderbilt University, Computer Science Department.

[Held 1970] M. Held and R. M. Karp. *The Traveling Salesman Problem and Minimum Spanning Trees.* Operations Research **18**:1138-1162, 1970.

[Held 1971] M. Held and R. M. Karp. *The Traveling Salesman Problem and Minimum Spanning Trees: part II.* Math. Programming **1**:6-25, 1971.

[Johnson 1990] David S. Johnson. *Local Optimization and the Traveling Salesman Problem.* Automata Languages and Programming: 17th International Colloquium Proceedings. 1990.

[Jones 1995] Antonia J. Jones and Christine L. Valenzuela. *Evolutionary Divide and Conquer (II).* (In preparation).

[Karp 1977] R.M. Karp. *Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane.* Mathematics of Operational Research vol.2 no.3. August 1977. 209-224.

[Lin 1965] S. Lin. *Computer solutions of the traveling salesman problem.* Bell system Tech. J. 44, 2245-2269.

[Martin 1992] O. Martin, S. W. Otto, and E. W. Felten. *Large-Step Markov chains for the TSP in cooperating local search heuristics.* Operations Research Letters, **11**(4):219-224, 1992.

[Or 1976] I. *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking.* Unpublished Ph.D thesis, Northwestern University, Evanston, IL, 1976.

[Valenzuela 1994] Christine L Valenzuela and Antonia J. Jones. *Evolutionary Divide and Conquer (I): a Novel Genetic approach to the TSP.* Evolutionary Computation 1(4): 313-333. MIT Press 1994.

[Valenzuela 1995a] Christine L. Valenzuela. *Evolutionary Divide and Conquer: a novel genetic approach to the TSP.* Ph.D Thesis,Department of Computing, Imperial College of Science, Technology and Medicine, London. (In preparation).

[Valenzuela 1995b] Christine L. Valenzuela and Antonia J.Jones. *Estimating the Held-Karp lower bound for the geometric TSP.* (Submitted to the European Journal of Operational Research, January 1995).