Proceedings of the 6th AGILE
April 24th-26th, 2003 – Lyon, France

THE JPATHFINDER MULTICRITERIA PATH PLANNING TOOLKIT

Peter Mooney and Adam Winstanley

Intelligent Graphical Data Research Group, Department of Computer Science, National University of Ireland Maynooth, Co. Kildare. Ireland. Telephone : +353 1 708 6099
Email : pmooney@cs.may.ie, Adam.Winstanley@may.ie

1. INTRODUCTION

The design, software development, implementation and end-user methodologies of a Java driven software toolkit (JPathFinder) for multicriteria path planning is described. This toolkit may be used to help decision makers (DMs) find the multicriteria paths (the Pareto Optimal set) that are best suited to a specific multicriteria route (or path) planning problem they are trying to solve. This paper reports current progress in the research and development work currently being carried out on JPathFinder. JPathFinder, developed within the evolutionary computation paradigm, generates a Pareto optimal set of nondominated paths. DMs have full-parameterised control over optimisation criteria specification, examination of interesting solutions, and JPathFinder path-search stopping conditions. Data clustering techniques are incorporated offering DMs greater control over the size of their output set. JPathFinder delivers platform independence and interoperability.

1.1 Path Planning Computation

Spatial multicriteria decision problems typically involve a set of geographically defined alternatives (events, paths, objects) from which a choice of one or more alternatives must be made in respect to a set of evaluation criteria [1]. An explicit geographical component is included making spatial multicriteria analysis different from conventional MCDM (multicriteria decision making) procedures. Therefore spatial multicriteria analysis can be seen as a three-stage hierarchical process of pre-processing, solution design and finally solution recommendation. Problem and environmental data is acquired in the pre-processing stage. Problem data may be expressed as a set of equations or constraints and the environmental data itself is usually contained in flat files or tabular data formats. Solution design involves formal algorithm and data structures modelling in order to generate a set of candidate solutions representing feasible spatial decision alternatives. Solution design must provide path-planning services with three key functions: path computation, path evaluation and finally path display.

The goal of path computation is to locate a connected sequence of network edges or links from a current location node to a destination node. *Path computation* may be based on a single or multiple criteria. Single criteria path computation is realised with any of the classical *shortest path* such as Dijsktra's or Bellman Ford algorithm (for example). When the number of criteria involved is greater than one, the problem cannot be solved by these standard approaches. Many applications form a linear combination of all criteria and optimise the resultant cumulative value. While such approaches admit solutions in polynomial time they do not adequately represent the multidimensional nature of the problem. The linear combination approach can only account for linear relationships among the criteria. In Horn[2] he states that as a result these aggregation approaches are open to criticism as it is generally impossible to combine conflicting criteria into a single criterion

prior to search. Many users and researchers now chose to first apply search to find a set of "best alternatives".

Path evaluation finds all of the attributes of a given path between the start and end nodes. Path evaluation is required for both the optimisation algorithms and to output the solution set to the DM. The goal of *path display* is to effectively communicate the set of optimal paths to the DMs. In this paper, our attention is focused strictly on *path computation* and *path evaluation*. Path display is important to facilitate more insightful decision making with the assistance of helpful graphical representations of Pareto Optimal subspaces. Providing appropriate functionality in order to illustrate a set of alternative criteria vectors is beyond the current scope of JPathFinder. Miettinen [3] provides a summary of a number of alternative means to illustrate the Pareto Optimal Set using bar-charts, petal diagrams and scatterplots.

1.2 Multicriteria Path Optimisation

Multicriteria path optimisation addresses path planning problems with several, often conflicting, cost criteria. For example, multicriteria path planning may involve optimising criteria including, for example, overall path distance, minimising the number of turns in a path and travel costs. For many years, researchers have been actively investigating methods to generate satisfactory computationally efficient solutions to these types of path planning problems. In contradiction to traditional optimisation methods combining all objectives into a linear, weighted, sum parallel optimisation, methods such as Evolutionary Algorithms (from Evolutionary Computation) allow direct convergence to the set of *nondominated solutions*. In fact, evolutionary algorithms are readily modified to deal with multiple criteria by incorporating the concept of Pareto Optimality. A standard introduction to the theory of Pareto Optimality is provided in Zitzler [4]. This allows DMs to choose *a posteriori* from a set of Pareto-optimal solutions *evolved* after a series of evolutions (or iterations) of the Evolutionary Algorithm. Several types of evolutionary strategies have been developed to solve multicriteria problems in many different application domains (outlined in Horn [2]), for example, industrial engineering applications [5], environmental management [6], and route generation in aircraft routing [7].

Evolutionary computation encompasses a host of methodologies inspired by natural evolution that are used to solve hard problems. As evolutionary algorithms possess several characteristics that make them well suited to these types of problems, evolution-based methods have been used for multicriteria optimisation for more than a decade. Evolutionary multicriteria optimisation has become established as a separate sub-discipline combining the fields of evolutionary computation and classical multiple criteria optimisation. The evolutionary approach implemented in JPathFinder is a cross fertilisation of methodologies derived from algorithms described in Horn and Zitzler [2,3] and Costelloe [8]. The overall system component outline of JPathFinder is illustrated in Figure 1.

2. DEALING WITH MULTIPLE CRITERIA

As outlined above there are several ways one can deal with multiple criteria when designing a solution to a given multicriteria path-planning problem. Morris [9] highlights the requirement, in some problems, for the DM to explicitly express their preferences by assigning weights to the selection criteria. He identified this as one of the major drawbacks of existing GIS/MCDM systems. This is difficult, if not impossible, for a DM to do, especially as the number of criteria increases. Feasible "real-world" solutions are compromise solutions, resulting from trade-offs between various conflicting criteria. Thus they do not maximise single criteria, but rather find an efficient and acceptable balance between the requirements of the problem solvers and the resources available to them.

The traditional methods used to deal with de-facto multiple criteria problems are based on the idea of converting a multicriteria problem into a single criteria problem by summing up weighted criteria. Two serious drawbacks of this approach are that it does not allow user-controlled examination of interesting (driven by personal preference) Pareto optimal solutions and consequently weights can be counter-intuitive making it difficult to facilitate the generation of solutions having those properties. After this, DMs require an interactive tool to run until they find a solution best meeting their expectations.

The network data structure (G) used in the implementation is the classical G = (V,E), where V is the set of all nodes in the network and E the set of all edges connecting pairs of nodes in the set V. Multiple costs or criteria in the network are stored (with order maintained) using `Vector` objects for each edge in E. We say that the network G has a dimension C if every `Vector` on every edge has a size = C. If C = 1 then the problem reduces to the classical one-dimensional case solvable with, for example, Dijkstra's Algorithm or A*. A basic programmatic requirement when dealing with multiple criteria is considering how those criteria are represented. The criteria must be represented by a well-defined data structure that can be efficiently manipulated by all components of JPathFinder. All criteria are represented as `Long` integer numbers. `Float` or `Double` number types can be easily facilitated. The criteria on every edge are then stored in a `Vector` data structure. We use the term *path-description vector* (as used in Costelloe [8]) for the vector of criteria stored on every edge. With this vectoral representation there is no predefined limit on the number of criteria the graph data structure can model.

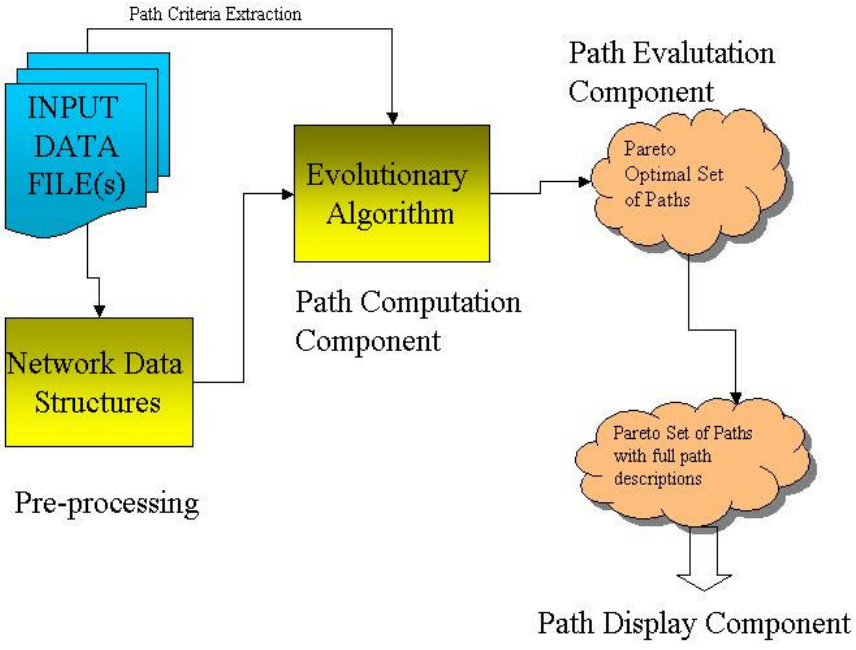

Fig. 1 The components of JPathFinder

2.1 Criteria Selection

JPathFinder is designed to deal with any number of path criteria. However DMs usually consider a smaller subset of criteria when dealing with multicriteria path problems. For example DM1 may consider {path distance, travel time and number of turns in the path} as their subset of criteria while DM2 may consider only {travel time and number of turns in the path}. With JPathFinder, DMs can choose to simultaneously optimise all criteria available or choose their subset of criteria thereby ignoring all others. Distance is by default an optimisation criterion. A literature search yielded other default criteria for inclusion. Timpf's [10] surveys of citizens in Zurich found people chose, in no particular order, minimal path time, minimal journey distance, least expensive paths, minimising the number of nodes passed through and finally the least complex path in terms minimising the number of transportation changes required (in the case of public transportation). Makaness [11], performing the same for Edinburgh, uses a route filtering mechanism to reduce the number of candidate paths for display by eliminating those with significantly high numbers of stops (stopping nodes). Their survey found that people prefer longer routes with fewer changes. From this the default minimisation criteria available in JPathFinder were derived as follows:

- overall path distance;
- overall path traversal time (if available);
- cumulative path expense (financial, fuel, other);
- total number of nodes.

JPathFinder generates a large number of non-dominated (Pareto Optimal) paths between pre-specified start and end nodes. A high number of solutions may be present in many path-finding problems. Data Clustering (as described in Zitzler [3]) may optionally be used to help DMs discover the nature of their trade-off solutions and be more informed when exploring the alternative paths presented to them. Clustering reduces the number of Pareto Optimal Solutions presented without destroying the characteristics of the Pareto Tradeoff Front. Thus only solution paths sufficiently *distinct* from one another are accounted for as output path solutions.

3. CLUSTERING THE PARETO OPTIMAL SOLUTION SET

The Pareto Optimal Set in problems such as multicriteria path planning can become extremely large and contain more solutions than a DM can actually evaluate in a straightforward and efficient manner. A key principal of Pareto optimality is that given a set of Pareto Optimal solutions no solution exists which could decrease one criteria without causing a simultaneous increase in at least one other criterion. This could be restated as that in the Pareto Optimal Set no solution is *better* than another – all members of the set are equally valid for implementation by the DM. The process of reducing the set is not just a simple case of *dropping* certain members of the set or randomly selecting a subset and discarding the remainder.

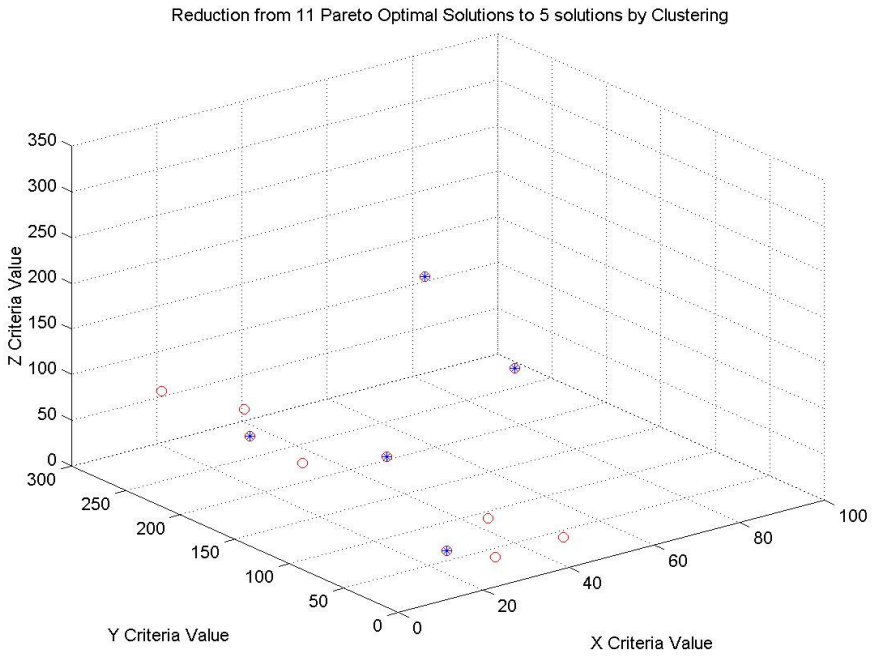

Fig. 2 11 Pareto Optimal Solutions with 5 Clusters Chosen

Most DMs want to simply arrive at a final solution to their problem and are not interested in exploring the entire Pareto Optimal Criteria space. From this viewpoint, being presented with all Pareto optimal solutions computed is of little use, especially when the cardinality of the Pareto Optimal Set is large. It would be useful for the DM to outline what they deem as a reasonable bound on the cardinality of the Pareto Optimal Set that they are presented with. However there is no way before the evolutionary algorithm starts of estimating how the Pareto Optimal Set will be distributed in criteria space. Figure 4 shows 9 Pareto Optimal Solutions in 2-D criteria space. The Pareto Front is clearly recognisable. If there is poor distribution among candidates in the set then many *similar* solutions are included in the set. For example the vectors p1 {40,50} and p2 {42,52} may be deemed *similar* due to their close proximity in criteria space. The final goal for the optimisation component after computing the Pareto Optimal Set is to only prune the given Pareto Optimal Set into one of manageable size. This pruned set should be a representative subset maintaining the important characteristics of the original Pareto Optimal Set. In order to perform this pruning, Cluster Analysis is performed.

3.1 JPathFinder's Clustering Technique

Cluster Analysis partitions a collection of n objects into k groups or partitions of relatively homogeneous elements (k < n). There are a large number of clustering algorithms reported in the literature [12]. Generally speaking, clustering algorithms can be classified into four groups: partitioning methods, hierarchical methods, density-based methods and grid-based methods. For JpathFinder, hierarchical clustering was chosen. Initially in hierarchical clustering (as outlined in Zitzler [4]) the whole Pareto Optimal set forms a basic cluster. Then iteratively two clusters are chosen to partition each larger one until the given number of clusters (k) (as chosen by the DM in advance) is reached. The clusters are

chosen by the nearest neighbour criterion where distance between clusters is given by the average distance between pairs of Pareto Optimal solutions across the two clusters. Selecting a representative individual for each cluster forms the reduced Pareto set.

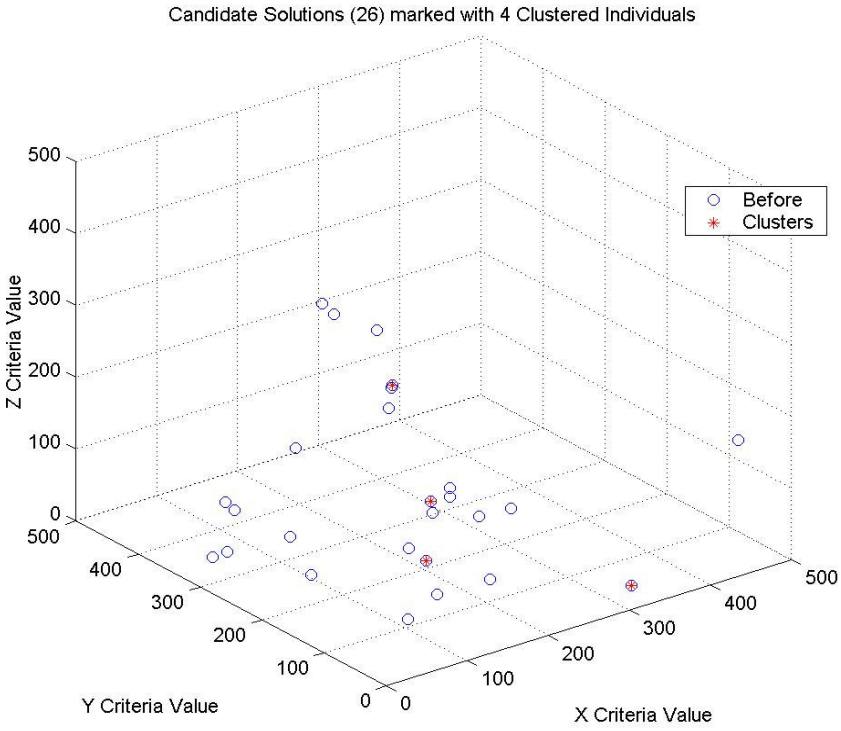

Fig. 3 Pareto Optimal Set with Cardinality 26 and 4 clusters

An illustration of this clustering technique is outlined in Figures 2 and 3 where the whole Pareto Optimal Set is plotted in criteria space with the chosen clustered individuals also marked. In the case of Figure 2, a cluster size of 5 was chosen and in Figure 3 a cluster size of 4 was chosen. The representative solution chosen in each case is the *centroid* – that is the point with minimum average distance to all other points in the cluster. The *centroid* solutions are clearly evident in Figure 2. In both cases the Cluster centroids are heavily shaded.

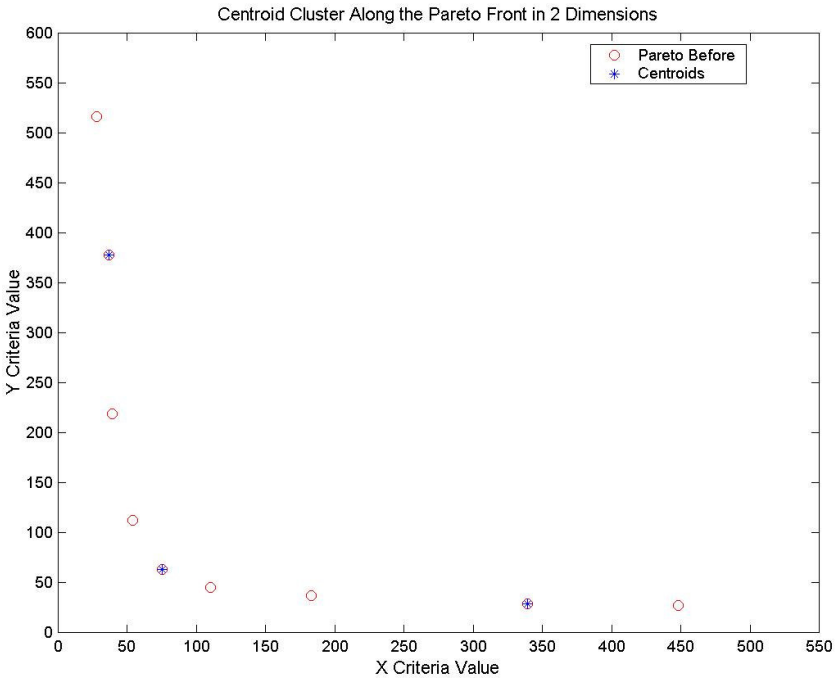

Fig. 4 Pareto Solutions and Cluster Centroids on 2-D Criteria Space

4. IMPLEMENTATION ISSUES

Java was chosen as the implementation language for JPathFinder (the 'J' being derived from Java). Moreira [13] comments that until recently there were few areas of Java so blatant performance deficient as that of numerical computing. Initial experiences of many developers of high performance numerical applications have seen them reject Java out-of-hand as a platform for their applications. Once it is accepted that performance is only an artefact of particular implementations of Java, there are no technical barriers to Java achieving excellent numerical performance. Using the Object Oriented paradigm of Java, the principal data structure components of JPathFinder are *objects*. Through this highly object-oriented approach to the development of this path planning toolkit, the task of re-implementation in another high level programming language (such as C++) should be straightforward for experienced programmers. This solves any interoperability problems as encountered with some other path planning software packages written in other 3rd generation programming languages.

The principal configuration and search set-up parameters in JPathFinder are as follows:

- Input files are in the standard graph-network format DIMACS Implementation Challenge network files (see Goldberg [14] where this format was first introduced). DIMACS facilitates a standard file format for graph input files that is flexible for many types of graph and network problems;
- There are many ways to internally represent path solution individuals (binary strings, real numbers, parse trees) -JPathFinder uses `String`;

- A configuration file is used to store all user options and parameters, before JPathFinder starts the evolutionary algorithm, as a means of reference afterwards;
- An archive of all paths considered Pareto Optimal at any stage of evolution is maintained and is available if requested by users;
- Users must supply the following parameters:
  - Start and End nodes of their target path;
  - Number of generations allowed for the evolutionary algorithm (Defaults to 10);
  - The rate of `crossover()` and `mutation()` of the evolutionary algorithm (Default values are included provided);
  - The number of candidate paths generated on every generation (Defaults to 20).
- With the incorporation of data clustering users may specify the number of Pareto Optimal Paths (hence specifying the number of clusters) they wish to be presented with in the final output set when JPathFinder has stopped.
- The system can be started in default mode – where the user only inputs the start and end nodes of their path.

5. REFERENCES

[1] Malczewski, F. A GIS-based Approach to Multiple Criteria Group Decision Making. *International Journal of Geographical Information Systems.* Vol 10. Issue 8. 955 – 972. 1996

[2] Horn, J. Multicriteria Decision Making and Evolutionary Computation. *The Handbook of Evolutionary Computation.* July 1996.

[3] Miettinen, K. Nonlinear Multiobjective Optimisation. *Kluwer Academic Publishers*, Boston, USA. 1999.

[4] Zitzler, E. and Thiele, L. Multiobjective Evolutionary Algorithms : A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation.* vol 3. Issue 4. 257 – 271. 1999

[5] Triantaphyllou, E. and Evans, G. Multi-criteria Decision Making in Industrial Engineering, *Computers And Industrial Engineering.* vol. 37. number 3. 505 – 506. 1999.

[6] Beinat, E.,Multi-criteria analysis for environmental management. *Journal of Multi-Criteria Decision Analysis*. vol 10. number 2. 2001

[7] Oussedik, S., Delahaye, D. and Schoenauer, M. Flights Alternative Route Generator by Genetic Algorithms. *Proceedings of the 2000 Congress on Evolutionary Computation.* IEEE Service Center, Piscataway, New Jersey. 896 – 901. 2000.

[8] Costelloe, D., Mooney, P. and Winstanley, A. An Evolutionary Spatial Decision Support System. In *Proceedings of GISRUK 2002,* University of Sheffield, UK. 91 - 94. 2002

[9] Morris, A. and Jankowski, P. Combining Fuzzy Sets and Databases in Multiple Criteria Spatial Decision Making. *Flexible Query-Answering Systems*. 103-116.

[10] Timpf, S. and Heye, C. Complexity of Routes in Multi-modal Wayfinding. *Proceedings of the 2nd International Conference on Geographic Information Science.* Boulder, Colorado, USA. 2002.

[11] Rainsford, D. and Mackaness, W. Mobile Journey Planning for Bus Passengers. *Proceedings of the 2nd International Conference on Geographic Information Science.* Boulder, Colorado, USA. 2002.

[12] McQueen, J. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkley Symposium on Mathematics, Statistics and Probability.* 1967.

[13] Moreira, J.E., Midkiff, S.P., Gupta, M., Artifas, P.V., Wu, P. and Almasi, G. The NINJA Project : Making Java work for High Peformance Computing. *Communications of the ACM*. Volume 44. Issue 10. 102 – 109. 2001.

[14] Goldberg, A.V. Scaling Algorithms for the Shortest Path Problem. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms.* Austin, Texas. 222 – 231. 1993.

[15] Zitzler, E. and Thiele, L. (1998b). An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. (Technical Report No. 43). Zurich: Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology.