Skill-based Resource Allocation using Genetic Algorithms and Ontologies

Kushan Nammuni[1], John Levine[2] & John Kingston[2]

[1]Department of Biomedical Informatics,
Eastman Institute for Oral Health Care Sciences, University College London, U.K
E-mail: nammuni@hotmail.com
[2] Artificial Intelligence Applications Institute,
CISA, Div. Of Informatics, University of Edinburgh, U.K
E-mail: John.Levine, John.Kingston@aiai.ed.ac.uk

Abstract Allocation of tasks to resources is a crucial issue for an organisation. This paper investigates the use of genetic algorithms, combined with an ontology, in producing optimal allocations of tutors to tutorials. The ontology is used to support partial matching; it helps determine which tutors' skills (i.e. resources) are ontologically close to the optimal skill. These are then matched against the skill requirements of the task. The other novel feature of this GA implementation is the steady state duplication elimination algorithm, which (when coupled with 10% best chromosome crossover) produces more solutions with better fitness than a standard steady state GA.

1. Introduction

Project planning is a notoriously difficult task. Typically, a project planner has to cope with all the complexity of combinatorial optimisation problems, along with restricted resources. Where the general problem is to allocate known tasks to limited resources, the problem is effectively a scheduling/timetabling problem [1]. In order to make good allocations in such situations, some method of rating a candidate allocation of resources to tasks, and some method of reasoning about these ratings are needed.

The work described in this paper investigates the use of a genetic algorithm in producing optimal allocations of tasks to resources, using an ontology to provide simple resource ratings. The domain chosen is tutor allocation i.e. the allocation of teaching resources (tutors/demonstrators/teaching assistants) to teaching tasks (tutorials). The tutors have various skills that make them more or less appropriate for different tutorials. For example, a tutor with excellent skill in C++ would be ideal for a C++ tutorial, acceptable for a Java tutorial (since there are similarities between C++ and Java) but of little use for a tutorial on natural language processing. The specific task chosen is the allocation of tutors to tutorials in the School of Artificial Intelligence at the University of Edinburgh, where 40 tutors must service 80 tutorial groups (each tutorial is repeated for different groups of students) that require 14 separate skills. The following entities and relationships were identified for this specific task:

- ***Tutors*** – defined by their knowledge, preferred time and number of tasks, rate per hour and availability.

– *Knowledge (1-5)* – the person's overall knowledge or expertise for a particular skill. Each tutor has a "knowledge score" for each skill of 1-5, corresponding to *aware, familiar, skilled, expert* and *guru*;

– *Rate (integer)* – the hourly rate is the employee's/tutors minimum wages per job. More experienced tutors earn higher rates;

– *Preferred time (morning/afternoon)* – this defines the preferred time of the day that the employee wants his/her tutorial sessions;

– *Preferred number of tasks (1-5)* – a tutor can define a preferred number of tasks/tutorials that he wants in a weekly schedule. The default is 2.

– *Availability (pair of values – day, morning/afternoon)* – specifies those mornings and afternoons each week when a tutor is (un)available.

- ***Module / Tutorials*** – defined by skill requirements, tutorial slots and budget.

– *Skill requirement (0-5)* – defines the skills needed to tutor the particular module;

– *Slots (pair of values: day, morning/afternoon)* – defines the timetable slots available to tutorials;

– *Budget (integer)* – specifies the allocated salary budget per module.

2. Knowledge Representation

Tree Structure	Node Type	Level
♦ **Programming**	*Parent*	*1*
▪ AI_programming	*Parent*	*2*
• *Lisp*	*Leaf*	*3*
• *Prolog*		
▪ General_programming	*Parent*	*2*
• *C++*	*Leaf*	*3*
• *Java*		
♦ **AI_techniques**	*Parent*	*1*
▪ Knowledge representation	*Parent*	*2*
• *Fuzzy Logic*	*Leaf*	*3*
• *1st ord. Pred.Calculus*		
▪ Search	*Parent*	*2*
• *Evolutionary Comp.*	*Leaf*	*3*
• *Constraint Satis. Prob.*		
• *Planning*		
▪ Learning	*Parent*	*2*
• *Neural Nets*	*Leaf*	*3*
♦ **Cognitive_science**	*Parent*	*1*
▪ *Linguistics*	*Leaf*	*2*
▪ *Psychology*		
▪ *Human Computer Interac.*		
♦ **Research Techniques**	*Parent*	*1*
▪ *Experimental Methodology*	*Leaf*	*2*

Figure 1 - Ontology of skills.

Figure 1 illustrates the ontology of skills that was used in this work. Leaf nodes represent the skills available or required; parent nodes represent the technical area classifications. This ontology is specific to Artificial Intelligence topics that were taught at the University of Edinburgh in 1999/2000.

In order to represent the relationships between leaf nodes, a weight is allocated to each pairing. The weight represents the ability of a tutor to undertake different tasks even if the tutor does not possess the exact job requirements. The weights are allocated on a "family tree" basis: siblings (i.e. two skills with the same parent) are awarded a relative weight of 0.25; first cousins are awarded a relative weight of 0.5. For example, C++ and Java would have a relative weight of 0.25, while Java and Prolog would have a relative weight of 0.5.

Each tutor is assigned a skill level in one or more of these fourteen areas, and required skill levels are also assigned to taught modules. For example, the *Learning from Data* module requires basic knowledge (1) in *Java* and in *Evolutionary Computation* but very good knowledge (4) in *Neural Nets*. The advantages of this ontology-based approach are that it allows the possibility of graded partial matching (which is essential in a resource-constrained task) and that the hierarchical tree representation gives the user a good visualisation of the knowledge. For customisability, the ontology was used to generate a matrix of weights that can be edited by the user, thus allowing scope for preferences or exceptions.

3. Evolutionary Parameters

Algorithm: The main algorithm used is a variant of a standard genetic algorithm with tournament selection and one-point crossover. A novel aspect of this work was the combination of a duplicate elimination algorithm with a steady state GA; for details of other combinations that were considered, see [1]. A standard genetic algorithm works by generating new "chromosomes" (solutions) by mutating or combining existing chromosomes, adding these offspring to the original population, and then sorting the population according to a fitness function and removing the worst chromosomes to keep the population size constant. The steady state duplicate elimination (SSDE) algorithm [2] used on this project uses this method with one addition: at the point at which the worst chromosomes are removed from the population, duplicate chromosomes are also removed.

Encoding: Direct encoding has been chosen to represent the schedule in a chromosome [3, 4]. Each gene represents a possible assignment of a tutor to a tutorial group. The length of the chromosome is equal to the total number of tutorial groups. Tutors can be assigned to multiple tutorial groups, but tutorial groups can have only one tutor.

Selection mechanisms: Tournament [5] and rank based selection mechanisms have been used to select individuals for the genetic operations. A tournament size of $k = 5$ was employed; this is a rank based selection in which chromosomes in the population are sorted according to their fitness values, and then the first k chromosomes are selected for the best chromosome crossover method (see below).

Mutation: The intuition behind the two mutation operators (swap and random) is to introduce some extra variability into the population. A swap mutation [6] swaps two tutor allocations from randomly selected genes in the chromosome; this is the primary method used to introduce variability. Random mutation changes a randomly selected gene value (i.e. an allocated tutor) to another possible value (i.e. another tutor); this method generally has an adverse effect on fitness due to poor 'distribution' (some tutors now being over- or under-worked), but it can introduce ‘lost’ gene values back into the chromosome, thus helping to prevent premature convergence. A combination of these two methods shows better results than using them individually.

Crossover: The crossover operator is one of the most important genetic functions in evolutionary algorithms. This is a process where chromosomes exchange segments via recombination. We have investigated a standard one point crossover method and a “eugenic” best chromosome crossover method, in which we force a certain percentage of offspring to be generated from the chromosomes with the highest fitness values.

Fitness: The user is allowed to specify the relative importance of three fitness constraints: skill, finance (i.e. staying within budget) and distribution. The fitness function of a schedule is computed by assigning penalty points for the violation of each constraint. The current fitness function awards a low number of penalty points if tutors are overqualified for the skills required for a tutorial; a higher number of penalty points if tutors are have the right skills but are under qualified; and if a tutor has no skill in an area but does have a related skill, then a penalty point value is assigned that is modified by the ontological weight matrix. Penalty points are also assigned for going over the allocated budget; for tutors being allocated to a tutorial group at a non-preferred time; and for each tutor not given any tasks. Finally, a very high number of penalty points are awarded to any job conflicts that arise where a tutor is allocated to two tutorials simultaneously.

4. Results

Based on the School of AI example, our results show that the population size is directly proportional to the fitness value; a larger population size generates better schedules. However, increasing population size also produces a rapid increase in computation time.

The steady state duplicate elimination algorithm has tremendous benefits for this problem. As shown in Figure 2, the standard steady state algorithm produced a lot of duplicates and it prematurely converged into one solution. The SSDE algorithm, however, produced multiple solutions with better fitness values. The drawbacks of the SSDE technique are the overhead involved in carrying out a large number of equality tests whenever an offspring is created, and the reduced probability of reproduction of the best chromosomes due to duplicate elimination. The latter problem is addressed by use of the best chromosome crossover technique; after experiments with 5% and 10% best chromosome crossover, it was determined that forced generation of 10% of offspring from the best chromosomes gave the earliest convergence.

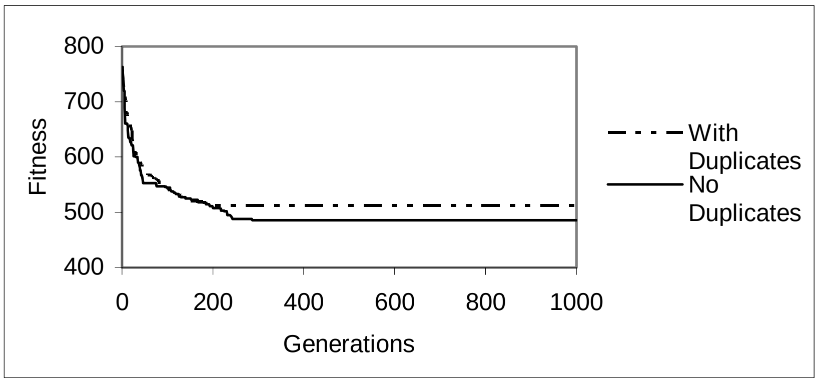

Figure 2 - Standard steady state Vs Steady state without duplicates

In order to compare the quality of the GA produced schedule, we solved a smaller version of the problem by hand. We employed several heuristic methods to allocate tutors to tutorial groups. Initially we allocated the most specialised subjects followed by others. In the assignment of tutors, the most important factor was to match skill requirements. The distribution of the tutors was the next factor considered; all the tutors were assigned to two tutorial groups in the same module. The finance factor was ignored to save time. Even after checking for timetable conflicts, it took 35 minutes to fulfil all the tutorial slots.

Table 1 illustrates the fitness values of the hand generated and GA produced schedules. It shows that individual penalty constraints are lower in GA schedule than in hand implemented schedule. Therefore, it proves that the GA performs much better than the hand produced allocation for the same elapsed time.

	Hand Created	GA Generated
Skill Fitness	589.0	499.5
Distribution Fitness	0.0	38.0
Finance Fitness	24.0	22.5
Pref. Time	29.3	17.3
No. Tutors Not allocated	0	0
No. Job Conflicts	14	0
Overall Fitness	3778.3	577.3
Total Time (Min)	35	35

Table 1 - Comparison between hand generated and GA produced schedules

5. Discussion

This approach seems promising for the general task of allocating tasks to resources. One possible improvement would be the use of multi-perspective [7] ontologies; for example, considering the similarity between skills based on a "how" criterion (i.e. skill A *is required for* skill B) rather than a "what" criterion (skill A *is in the same category as* skill B). Applying this approach to our domain, Prolog would be required for First Order Predicate Logic, while C++ might be considered a distant relative of Evolutionary Computation. Other ontologies are also possible, and an accurate relative weighting matrix should draw on several ontologies to derive its weights.

References

1. Hsiao-Lan Fang. *Genetic Algorithms in Timetabling and scheduling.* (Ph.D. thesis) Department of Artificial Intelligence, University of Edinburgh, 1994
2. Darrell Whitley. *GENITOR: a different genetic algorithm.* In Proceedings of the Rocky Mountain conference on Artificial Intelligence. Denver, Colorado.
3. M Gröbner, P Wilke. *Optimizing Employee Schedules by a Hybrid Genetic Algorithm.* E.J.W Boers et al. (Eds.) EvoWorkshop 2001, LNCS 2037, pp. 463-472. Springer-Verlag, Berlin, Heidelberg.
4. C. Di Stefano, A. G. B. Tettamanzi. *An Evolutionary Algorithm for solving the School Timetabling Problem.* E.J.W Boers et al. (Eds.) EvoWorkshop 2001, LNCS 2037, pp. 452-462. Springer-Verlag, Berlin, Heidelberg.
5. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd revised and extended edition. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 3-540-60676-9.
6. Hugh M Cartwright. *Getting the Timing Right – The Use of Genetic Algorithms in Scheduling.* Proc. of Adaptive computing and information processing conference. Brunel, London. pp 393-411 (UNISYS 1994)
7. Kingston J., *Multi-perspective modelling: A Framework for Knowledge Representation*. Forthcoming.