Tabu Search Articles summaries

William Bezuidenhout

Friday, 16 April 2010

Application of Self Controlling Software Approach to Reactive Tabu Search

Self adaptive software is software that changes itself at runtime to achieve better performance. Heuristic search algorithms are software which try to find good solutions to optimization problems within reasonable time limits. Reactive heuristics search algorithms modify algorithm paramters and/or strategies during the search to improve search quality.

Most of the real life combinatorial optimization problems cannot be solved with exact algorithms within reasonable time limits. Heuristic algorithms are used to solved these problems but the performance of these algorithms depends on the values of their paramters.

This paper uses the self controlling software paradigm proposed by Koar, Eracar and Baclawski and use a control theoretic approach to adjust the parameters of the controlled plant - the tabu search algorithm.

Laddage separated the work on self-adaptive software into three groups: coding as a dynamic planning system, coding as a control system and coding as a self aware system. In the coding as a dynamic planning system approach, a system plans its actions and re-plans when the effectiveness of the plan decreases. In the coding as a control system approach, the software is a system with monitoring and control units. The reconfiguration of the system is managed by the evaluation, measurement and control units. In the coding as a self aware system approach evaluation, revision and reconfiguration is part of the running software. The application has knowledge of its operation, evaulates itself, reconfigures and adapts to changes.

In engineering, *control theory* is used for controlling dynamic systems. Kokar, Eracar and Baclawski identified software as a candidate for a control

plant whose efficiency can increase by dunamic adjustments. This approach is called the *self controlling software approach.*

The control theory based self-controlling software model maps the conecpt of control theory to software engineering. In this model:

- Sofware is treated as the controlled plant.
- The bahavior of the plant and environment is modelled as a dynamic system.
- Inputs to the plant are classified as *control inputs* and *disturbances.* Control inputs control the behavior of the plant according to the control goal, while disturbances change the plant's behavior unpredictably.
- A *controller* subsystem changes the value of control inputs to the plant.
- A *quality of service* (QoS) subsystem provides feedback to control the plant.

Tabu Search (TS) is an iterative heuristic algorithm. TS uses the history of search (*memory*) to prevent cycling back to previously visisted solutions. In each iteration, a transformation operator - the *move* is used to generate the *neighborhood* of the current solution. The moves operated stored in the *Tabu list.* A move is *tabu* if it reverses one of the transformations on the tabu list. The tebu list size *ts* indicates the number of iterations a move will be considered as tabu. If the *aspiration condition* is met, a move is not prohibited. One commonly used aspiration condition is: if a move results in a solution which has a better objective value than the best solution, it is not tabu.

The intesification strategies direct tabu search to search for solution similar to each other whereas diversification drives the search to unexplored areas in the solution space.

Iterative search algorithms moves through space of solutions by selecting a different solution in each iteration. They use different intensification and diversification strategies to direct the search. Intensification refers to focusing the search efforts on a region within the solution space. Diversificatio, refers to driving the search to different regions. Finding a balance between intensification and diversification is important for the success of the search algorithms. The balance is determined by some paramters of the algorithm.

Even while solving a specific problem the need for intesification and diversification cahnge during the search depending on the region the search is currently in.

The control theory based self-controlling software model is used for controlling the intensification of the search. In this model the software is treated as a system whose parameters can be dynamically changed using a feedback controller. A feedback control system is composed of the following elements. The *Target system* (also referred to as *Plant*) is the system to be controled. In this case it is the tabu search. The goal is to control the intensification of the search. The *Controlled output* is a variable of the target system that is controlled. The *Reference input* is the desired value of the measured output. The *Controller* is an equation which determines the value of the control input, given reference input and the controlled output.

We performed some experiments with the tabu search algorithms to support our decision on the control input and output. In the experiements, we changed the value of tabu list size ts according to a schedule and observed the repetition length l_r.

It is seen that when ts decreases the repetition length decreases and increases when ts increases. The decrease of the repetition length to very small values menas that the search got trapped around a local optima. As ts values increas, l_r values increas and reaches to 100 which means the solutions found has not been met within the last 100 iterations. Having repetitions shows the search is intensifying on a region, whereas having no repetitions for a while shows that search si moving away from solutions found within the last 100 iterations.

The l_r values recorded during the experiments and it is seen that most of the existing repetitions occur withint the last 100 ieterations. Hence the size of the sliding window to measure the repetition length was selected as 100.

The elements of the feedback control system can be mapped to the tabu serach control system as follows: *Target system:* Tabu Search Algorithm; *Control input:* Tabu List Size; *Measure Output:* Repetition Length, l_r;*Reference Input:* The desired repetition length, l_r^{ref}; *Control Error:* The difference between the desired repeptition length and the measure output, $e = l_r^{ref} - l_r$

Intemsification alone is not sufficient for a sucessful search. The search should also be directed to unexplored areas of the search space, in other

words 'diversify'. Another mechanism was added to diversify the search. This mechanism gathers information about the history of the search and decides on when and how to diversify. The diversification mecahnism is composed of two components, the *Observer* which is responsible for observing the use of the moves and the *Diversifier* which modifies the ts as instructed by the Observer. The Obversver monitors if the *stagnation period* the given threshold. The stagnation period is defined as the period of iterations iwthout any improvement with respect to the current best solution. The value of the stagnation threshold is selected equal to number of possible moves, p_o. When the stagnation period length exceeds p_o the Observer counts the number of movies that have not been used since he beginning of the stagnation period.

In the beginning of the diversification period, the Diversifier gets information from the Observer. This information includes the iteration number when the stagnation started or the last diversification period ended, and the number of moves that have not been used since that iteration. THe Diversifier increases and adjusts ts in each iterations such that the moves that have been used since that iteration are prohibited. Diversification perdio length equals to the nubmer of not used moves. Although diversification period length equals to the number of not used moves. Although diversification mechanism targets the use of old moves during period., it ecreases the ts to a small value (one third of the problem size) for short periods so that the old moves and new moves are combined.

Analyzing the results it can be seen that for uniform random distributed problems both Re-Tabu and SC-Tabu(algorhitm discussed in this paper) have better performance than Ro-Tabu (Robuts Tabu Search). For real-life problems SC-Tabu shows better performance than Re-Tabu in all cases and has better performance than Ro-Tabu in three out of five cases.

The development of a multi-objective Tabu Search algorithm for continuous optimisation problems

Meta-Heuristic optimisation techniques have proved effective in solving complex, real -world optimisation problems with many local mnima, problems to which traditional, gradient-based methods are ill suited. Some significant methods to have emerged in recent years are Simulated Annealing(SA), Genetic Algorithms (GA), Evolution Strageties (ES) and Tabu Search (TS).

With the realisation that most real-world design problems involve com-

promies between conflicting objectives, efforts have been directed towards developing multi-objective optimisation algorithms. The first multi-objective GA was developed in 1985 and ressearch in this field has been very active. Similalry, multi-bjectve SA and ES algorithms have alos been developed. However, very little work has been done on multi-objective TS algorithms.

In this paper we present two multi-objective TS algorithms. The first, previously presented algorithms is a straightforward adaptation of a single objectvie TS algorithm for continuous problems. The second algorithms contains significant enhancements, including a novel design varaible selection scheme (to improve local search efficiency) based on path relinking strategies common in discrete optimisation algorithms.

The motivation for these developments in the TS algorithm. Such problems have high computational demands, are highly constrained, and cannot be solved by gradient-bsed methods due to the lack of gradient information and the large number of local minima.

In the work by Kellar a parametic study on a number of optimisation problems and showed the sensitivity of an objective function to each design variable was not consistent across the design space. Thus he showed that optimising on a dynamically changing subset of design variables, rather than the entire set of variables, led to a great optimiser performance improvement.

Inspired by Keller's workm we modified his strategy so that it could be applied to any continuous optimisation problem and incorporated it into our multi-objective TS algorithm.

There are two commonly used approachesto solving a multi-objective optimisation problem. The first reduces the multiple objectives to a single objective by generating a composite objective function, usually from a weighted sum of the objectives. The second approach to solving the multi-objective problem is to search directly for the entire Pareto-optimal[1] set. This can be achived in a number of ways and requires modification to existing single-objective algorithms.

Path relinking strategies are quite common in the discrete optimisation field. The underlying principle is that knowledge of the path (or direction) in design spce can be used to guide the search along that path, given that is may be better seach direction than a random one, in the hope that it will

[1]Given a set of alternative allocations of goods or outcomes for a set of individuals, a change from one allocation to another that can make at least one individual better off without making any other individual worse off is called a "Pareto improvement". An allocation is defined as "Pareto efficient" or "Pareto optimal" when no further Pareto improvements can be made (*http://en.wikipedia.org/wiki/Pareto_efficiency*).

yield futher good points.

In this paper, we present two multi-objective TS algorithms. The first, previously presented algorithm we call MOTS (Multi-Objectve Tabu Search). The second algorithm ,which contains the newly developed design variable scheme, we call PRMOTS (Path relinking Multi-Objective Tabu Search). The fundamental basis for both algorithms is the same. The two algorithms differ in their local search pattern, a key component of TS and a n area in which significant performance enhancements can be made.

The single-objective TS implementation of Connor and Tilley is used as a starting point for multi-objective variants. This uses a Hooke and Jeeves (H&J) local search algorithm coupled with short, medium and long term memories to implement search intesification and diversification as prescribed by Glover and Laguna.

Recently visited points are stored in the *short term memory* (STM) and are Tabu – the search is not allowed to revisist these points. Optimal or near-optimal points are stored in the *medium term memory* (MTM) and are used for intensification, focusing the search on areas of the search space with known good objective funtion values. The *long term memory* (LTM) records the regions of the search space which have been explored, and is used on diversification, directing the search to regions which are under-explored.

The strategy used to move in the authors algorithms accept both uphill and downhill moves – the next point is simply the "best" allowed point (or one of the Pareto-equivalent best points) selected from the candidate solutions.

In the Conor and Tilley's single-objective TS, the MTM is a bounded, sortd list of near-optimal solutions. As the concept of a single optimal point does not exist in multi-objective optimisation, we replace the MTM in our multi-objective TS variant by an unbounded set of non-dominated solutions produced by the search.

An unbounded MTM was chosen for two main reasons. First, for continuous problems the dieal Pareto set is inifinite and an unbounded set is a logical discretisation of this concept. While there are legitimate concerns regards the presentation of a large number of Pareto-optimal solutions to a designer as the result of an optimisation run, we feel it would be better to deal with these issues at the post-processing stage. It is worth pointing out that some GAs use bounded archivse of optimal points because this is consistent with the use of fixed size populations in those algorithms.

The orignal single-objective TS produced intensification points by using the MTM to generate points in the neighbourhood of good solutions. At search intensification, a point is chosen randomly from the *Immediate*

Memory (IM). The IM is continuously updated and points which become dominated by the addition of a new point are removed. Thus, the IM should always contain points which are on, or near to, the current Pareto-optimal front (stored in the MTM).

The single-objective TS restart strategy returns the search to the current best point in the MTM. As the MTM is now a set of Pareto-optimal points, we simply select one point at random from the set.

The authors emply a very simply constriant handling strategy: any point which violates any constraint is deemed to be Tabu and the seach is not allowed to visit that point. Thus, accepted solutions are limited to feasible space. We refere to this as a binary constraint – solutions are either feasible or not and there is no need to quantify the extent of constraint violation.

A futher benefit to running our algorithm on highly constrained problems is the local search pattern at the heart of TS, which reduces the likelihood of generating infeasible solutions.

The most computational expensive step in the optimisation procedure is the solutions evaluations (allowing for points that are Tabu or violate constraints) – assuming that objecive functions evaluations are expensive compared to the optimisation algorithm logic, a reasonable assumption for most real-world problems – and there is consuderavke scope for efficiency savings at this step.

In the design variable selection scheme for the PRMOTS algorithm the authors keep all the points without removing those that are Tabu and *without makign any funciton evaluations*. The authors then assess the potential for improvement in the objective functions for each design variable in the following way.

Taking the current set of Pareto-optimal points (stored in the MTM) as the reference set, the Euclidean distance in design space between these two points and every point in the reference set is cacluted, THe smallest of these distance is saved, and the design variables are ranked according to this distance.

In essesence, thsi selection scheme is only allowing the optimiser to vary those variables that will progress the search from current point to the reference set in the fastest possible way. This assessment is based soley on a measure of the Euclidean distance in design space.

The test functions used in this study are from the ZDT family of problems. They have $10 \leq n\_var \leq 30$ and the number of objectives $n\_obj = 2$, and they are designed to have Pareto-optimal sets which are difficult to

locate accurately by means of an optimisation algorithm.

The algorithm chosen for comparison was NSGA-II which has been shown to be one of the better multi-objective Genetic Algorithms and has a number of high-quality, open-source software implementations freely available. Although various studies have been published which show competitive algorithms outperforming it on the family of ZDT problems.

There are two main reasons for the choice of NSGA-II as the benchmark algorithm. First, the algorithm is very well known, due both to its inclusion in a number of free and commercial software packages and to its featuring in a range of published studies. Second, the authors ultimate goal in developing their TS algorithms is not absolute performance on a set of benchmark problems. Rather, our development is motivated by the requirements of real-world optimisation problems, for which authors believe their TS algorithms to be well suited.

Performance assessment on a set of well known problems against a well known algorithm then serves a validation of their approach and provides them with suggestions for improvments on their algorithms.

The results showed that the variable selection scheme gave a clear improvement in performance over the basic MOTS algorithm. Additionally, both algorithms performed comparably with NSGA-II. Overall, bnoth TS algorithms exibited greater performance variability than NSGA-II, but this was compensated by a greater likely improvement in the objective functions.

Based on these findings, it appears that out PRMOTS algorithm is competitive iwith NSGA-II and both could be expected to perform well on other multi-objective optimisation probelms. NSGA-II may be better chioce where low performance variability is desired. However, as the authors have argued in their paper and as reported by other researchersm the local-search components and the constrain handling flexibility of TS makes it partuclarly attractive for problems which may be highly constrinaed and we believe that PRMOTS is a good choise for these types of problems.