AIPS-02

Workshop Notes

Planning and Scheduling with Multiple Criteria

Toulouse, France, April 23, 2002

Sponsored by PLANET

the European Network of Excellence in AI Planning

http://planet.dfki.de/

Brian Drabble
CIRL

Jana Koehler
IBM Zurich

Ioannis Refanidis
Aristotle University

Program Committee

Mark Boddy, *Honeywell*

Yannis Dimopoulos, *University of Cyprus*

Patrick Doherty, *Linkoping University*

Alfonso Gerevini, *University of Brescia*

Hector Geffner, *Universidad Simon Bolivar*

Richard Goodwin, *IBM T.J.Watson Center*

Martin Mueller, *University of Alberta*

Nicola Muscettola, *NASA*

Karen Myers, *SRI International*

Alexis Tsoukias, *LAMSADE, Universite Paris Dauphine*

Ioannis Vlahavas, *Aristotle University*

Joachim Paul Walser, *I2 Technologies*

Contents

Preface

Most real-world problems demand the consideration of many criteria, such as plan duration, resource consumption, profit, safety etc, either separately, or in some combination. In the former case the plans are optimized for a single criterion and the other criteria are handled as constraints, whereas in the latter case the plans are optimized with respect to an arbitrary combination of the criteria. In many cases the criteria are in conflict and a trade off must be identified. For example, in a manufacturing domain the criteria may be to maximize the work in progress (to maximize the number of orders fulfilled) and minimize inventory (to minimize the amount of raw materials purchased) but to fulfill a large number of orders a large inventory must be kept. In addition to resolving conflicts several issues arise when taking into account multiple criteria, such as defining optimality, expressing preferences, aggregating the criteria, generating bounds and/or heuristic distance information, guiding search, pruning branches, trading off planning time and solution optimality, etc.

Dealing with multiple criteria is not a unique problem faced by researchers in AI planning and scheduling. Evaluating states and solutions based on multiple criteria is a problem occurring in other fields, in particular, combinatorial game search and multi-criteria decision making. Researchers in these areas have tended to address these related problems from a search or operations research perspective, respectively.

During the last few years significant improvements have been made in the capabilities of planning systems to the point that they are now capable of producing plans with hundreds of actions in a few seconds. While such performance is commendable, it has been achieved with very simple action descriptions that would have little applicability on real-world problems. We believe that it is the time to investigate ways of improving action descriptions and to handle reasoning with multiple criteria, an area that has been neglected for too long.

The workshop has several goals:

1. to review the current state of the art in reasoning with multiple criteria,
2. to initiate discussions within the AI planning and scheduling communities on how these problems may be addressed , and
3. to initiate the transfer of applicable techniques, insights and experiences from other communities such as Operations Research, Uncertainty and Game communities.

The organizers

Multicriteria Evaluation in Computer Game-Playing, and its Relation to AI Planning

Martin Müller
Department of Computing Science, University of Alberta
Edmonton, Canada T6G 2E8
mmueller@cs.ualberta.ca

Abstract

Games are a popular test bed for AI research. Many of the search techniques that are currently used in areas such as single-agent search and AI planning have been originally developed for games such as chess. Games share one fundamental problem with many other fields such as AI planning or operations research: how to evaluate and compare complex states? The classical approach is to 'boil down' state evaluation to a single scalar value. However, a single value is often not rich enough to allow meaningful comparisons between states, and to efficiently control a search.

In the context of games research, a number of search methods using multicriteria evaluation have been developed in recent years. This paper surveys these approaches, and outlines a possible joint research agenda for the fields of AI planning and game-playing in the domain of multicriteria evaluation.

Introduction

The need for multicriteria evaluation techniques in game-playing programs is not immediately obvious. All popular games have final outcomes that are scalar, be it win-draw-loss as in chess, the number of points in games such as Go or Awari, or the amount of money in casino games. In games that are simple enough to allow a complete analysis, the exact value of a game position can be computed by the minimax evaluation rule (von Neumann 1928; von Neumann & Morgenstern 1947). However, complex games such as chess or Go rarely allow a complete analysis. Evaluation problems arise quickly when a player's knowledge about a game is less than perfect. In place of an intractable complete analysis, games are usually analyzed by a deep but far from exhaustive search using a scalar-valued heuristic evaluation function.

Another source of complexity are games where the complete game state is not known to a player because of hidden information such as the cards in other player's hands in most card games, or because of chance events such as dice throws or cards drawn from a deck during a game.

In this survey we will look at several techniques that use multicriteria evaluation in games. We start by summarizing some basic facts about the structures used in multicriteria evaluation: vector dominance, as used in multiobjective evaluations, and general partially ordered sets. Vector dominance defines a specific kind of partial order, and in turn each finite-dimensional partial order can be represented by a vector with the same dominance relation that is used in multiobjective evaluation.

The main part of the paper consists of an overview of two topics that the author has worked on: the search method of partial order bounding, and a class of games called combinatorial games which are based on a partial order of game values. We also briefly survey other related work on multicriteria techniques in games.

The final, mostly speculative part of the paper discusses possible relations between the two fields of game playing and AI planning. How can multicriteria planning methods be used in game programs? And how can techniques developed for game tree search be used in multicriteria planning?

Background

In this section we describe the sum-of-features model for scalar evaluation, give definitions of multiobjective and partial order structures, and point out their close correspondence, at least in theory.

The standard scalar approach: weighted sum of features

In the standard model of computer game-playing, position evaluation is a two step process. The first step maps a game position to an abstract representation. A number of relevant attributes are computed and collected in a high-dimensional feature vector $\mathbf{v}$. Within such a vector, single features are usually of a simple type such as 0-1 (boolean), integer, or real. Given a feature vector $\mathbf{v}$ and an integer- or real-valued weight vector $\mathbf{w}$, a scalar-valued evaluation is computed as the weighted sum $eval(\mathbf{v}) = \sum w_i v_i$.

The weighted sum approach to evaluation has been very successful in practice. It has proven to be a useful abstraction mechanism, with many desirable properties, such as simplicity, and ease of use in efficient minimax-based algorithms. Furthermore, in some games there is a natural mapping of positions to a numerical evaluation, for example the expected number of captured pieces in Awari or the balance of territory in Go. In games that end in a simpler outcome such as win, loss or draw, a scalar evaluation can be interpreted as a measure of the relative chance of winning.

Despite the great success of the weighted sum approach to evaluation, the method has quite a few weaknesses, and many of the alternative methods discussed in the survey (Junghanns 1998) were designed to address such weaknesses. The main drawback of using a single number for evaluation is that information is lost. All kinds of features are weighted, added and compared, even those for which addition and comparison do not really make sense. Problem topics include unstable positions, long term strategic features, and close-to-terminal positions. For a detailed discussion see (Müller 2001b). It is therefore natural to consider richer evaluation structures, such as partial orders.

Partially Ordered Sets

Our definitions follow standard conventions. For more information see textbooks such as (Stanley 1997; Trotter 1992).

A *partially ordered set* or *poset* P is a set (also called P) together with a reflexive, antisymmetric and transitive binary relation $\leq$. The dual relation $\geq$ is defined by $x \geq y \iff y \leq x$. Two elements x and y of P are called *comparable* if $x \leq y$ or $y \leq x$, otherwise x and y are called *incomparable*. The relation $x < y$ is defined by $x < y \iff x \leq y \wedge x \neq y$, and $x > y$ is equivalent to $y < x$. A nonempty subset A of P is called an *antichain* if and only if any two distinct elements of A are incomparable.

Multiobjective Evaluation

The standard approach to evaluation in multiobjective search (Stewart & White 1991; Harikumar & Kumar 1996; Dasgupta, Chakrabarti, & DeSarkar 1996b; 1996a) uses a m-dimensional vector of scalar values from domains $Y_1 \ldots Y_m$. A partial order on such vectors is defined by the following vector dominance relation:

$$\mathbf{y} \leq \mathbf{y}' \;\Leftrightarrow\; y_i \leq y_i' \;\forall i \in 1, \ldots, m.$$

In partial order terminology, the poset defined by the vector dominance relation is the direct (or cartesian) product of the totally ordered domains, $Y_1 \otimes \ldots \otimes Y_m$.

While it is clear from the definition that each multiobjective evaluation is a partial order, the converse is also true, in the following sense: Any poset P of finite *dimension* $dim(P)$ can be represented as the direct product of $dim(P)$ total orders (Ore 1962). However, it might be intractable to find such a representation for a given poset. Current algorithms for constructing a multiobjective representation for a given partial order are practical only for posets of "modest size" (Yanez & Montero 1999).

Any partial order technique can immediately be used in a multiobjective framework, whereas multiobjective techniques that rely on the vector dominance in their algorithm only work for the general case if a suitable vector representation can be found.

Approaches to Multicriteria Evaluation in Computer Game-Playing

In this section we investigate problems of combining partial order evaluations with minimax search. We discuss how partial order evaluations arise in games, and some algorithms for dealing with them.

Goals of Partial Order Evaluation The main goal of partial order evaluation is to make comparisons between positions only when they are meaningful. In contrast, standard scalar evaluations are applied and used to compare positions regardless of whether the underlying positions are comparable. By refraining from judgment in doubtful cases, partial order evaluation aims at increasing the confidence in the validity of *better* and *worse* judgments derived by search.

The Fundamental Problem of Using a Partial Order Evaluation in Minimax Tree Search

When using partially ordered evaluations, the result of a minimax search cannot be just a single value from the partially ordered set, because computing minima and maxima of such values is an ill-defined problem. A totally ordered set such as the integers or reals is closed under the application of the operators *min* and *max*: if $x_1, \ldots, x_n$ are values from a totally ordered set T, then both $\min(x_1, \ldots, x_n)$ and $\max(x_1, \ldots, x_n)$ are again elements of T, with the properties $\min(x_1, \ldots, x_n) \leq x_i$ and $\max(x_1, \ldots, x_n) \geq x_i$ for all $1 \leq i \leq n$. Furthermore, the minimum and the maximum coincide with one of these values. For values from a partially ordered set, it is no longer possible to define a *min* or *max* operator with these properties.

Solutions for Special Cases

Several different approaches to overcome this fundamental problem have been tried. In some special cases, it is possible to define meaningful *min* and *max* operators with similar but more restricted properties. One possible approach in the case when the poset is a lattice is to define the least upper (greatest lower) bound of a set of incomparable values as the maximum (minimum) of these values. (Ginsberg 2001) develops such a search model and applies it to the imperfect information game of Bridge. For general lattice-valued evaluations, that do not possess the special semantics used in Ginsberg's model, this approach loses information, since propagating such bounds by a tree backup leads to computing bounds of bounds of bounds etc., which makes the approximation weaker and weaker.

Another approach (Dasgupta, Chakrabarti, & DeSarkar 1996b) keeps track of a set of nondominated sets of outcomes. This can lead to high complexity in the case when there are many incomparable values. A viable search procedure seems to require extra assumptions, such as totally ordered private preferences of the players, so this approach does not seem to be used in practice.

Approximating Scalar Values

One natural way in which partial orders arise in game state evaluation is uncertainty about the true value of a scalar value. Intervals, "fat values" such as triples containing a lower bound, a realistic value and an upper bound (Beal 1999), and probability distributions (Baum & Smith 1997) are prominent examples. Different kinds of partial orders

can be defined over such structures. One natural interpretation of an interval is as a pair of upper and lower bounds on the unknown true value. The corresponding partial order is a vector dominance order. An example of a relatively strong ordering of probability distributions is *stochastic dominance* (Russell & Norvig 1995).

(Lincke 2002) studies the problem of building an opening book for a game, that can contain both exact and heuristic minimax scores. He defines a new type of min and max operators for "fat values" that keep value representations compact yet can preserve some information about choices between exact and heuristic plays. He further extends the model to deal with draw values arising from position repetition.

Partial Order Bounding (POB)

Partial order evaluations are useful since they are more expressive than scalar evaluations. One practical problem is that many partial order search methods try to back up partially ordered values through the tree. Depending on the method used, this leads either to potentially huge sets of incomparable options, or to a loss of information, or both. In addition, some methods are applicable only to restricted types or specific representations of partial orders. *Partial order bounding (POB)* (Müller 2001b) avoids such problems by separating the comparison of partially ordered evaluations from the tree backup.

Partial order bounding is based on the idea of null window searches, which have already become very popular with scalar evaluation through search methods such as SCOUT (Pearl 1984) and MTD(f) (Plaat *et al.* 1996). Rather than directly computing minimax values, null window searches are used to efficiently compute bounds on the game value. In SCOUT, the purpose is to prove that other moves are not better than the best known move, while in MTD(f) the minimax value is discovered by a series of null window searches. The goal of a single null window search is to establish an inequality between a given fixed bound and the evaluation of a node in the search tree. POB extends this idea to the case of partial order evaluation.

In the case of a poset P, a bound $B \subseteq P$ can be given by an antichain in P that describes the minimal acceptable outcomes a player wants to achieve.

The *success set* of B in P is defined by $S(B) = \{x \in P \mid \exists b \in B : x \geq b\}$, and the *failure set* of B in P is the complement of the success set, $F(B) = P - S(B)$.

The success set contains all values that are "good enough" with respect to the given bound B, while the failure set contains the remaining insufficient values. Minimax search is used to decide whether the first player can achieve a result $x \in S(B)$, or whether the opponent can prevent this from happening. In the example tree shown in Figure 1, leaves have been evaluated by pairs of integers. The usual vector dominance order is used. In the diagram, squares represent MAX nodes and circles MIN nodes. For illustration, we consider the following two out of the large number of possible bounds: $B_1 = \{(5,7),(10,3)\}$ and $B_2 = \{(5,8),(6,4)\}$. In this example, MAX can obtain the bound B_1 but fails to obtain the bound B_2. Leaf evaluations and backed-up values are shown in the figure, with a plus sign representing success and a minus sign representing failure for MAX. Note that MAX would not succeed by selecting one of the two single-element subsets of B_1 in this example.

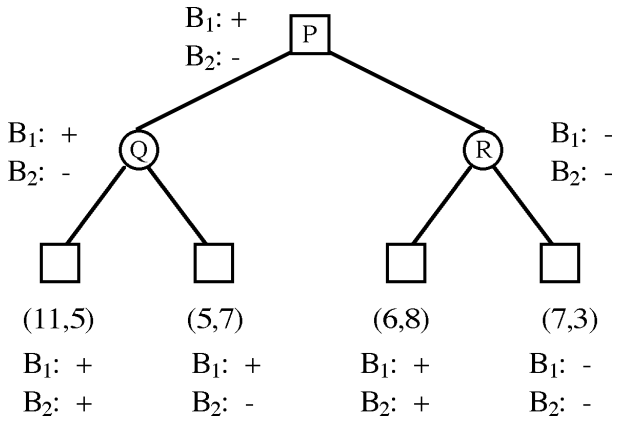


Figure 1: Example of search using POB

In POB, the comparison of partially ordered values is separated from the value backup procedure in the game graph. This simplifies the computation compared with previous approaches, since there are no sets of incomparable values that must be computed, stored, and backed up.

Partial order bounding can be combined with any minimax-based search method, such as alpha-beta or proof-number search (Allis 1994). A partial order evaluation can be added to a sophisticated state of the art search engine with minimal effort. The method has been successfully applied to solving capturing races in the game of Go (Müller 2001b).

The next topic, decomposition search, represents a very different approach to minimax game analysis, which leads to a partial order evaluation as well.

Combinatorial Games and Decomposition Search

Decomposition search (Müller 1999) finds minimax solutions to games that can be partitioned into independent subgames. The method does not use traditional minimax search algorithms such as alpha-beta, but relies on concepts from combinatorial game theory to do locally restricted searches. The result of each local search is an element from the partially ordered domain of combinatorial games (Conway 1976; Berlekamp, Conway, & Guy 1982). In a last step, combinatorial games are combined to find a global solution. This divide-and-conquer approach allows the exact solution of much larger problems than is possible with alpha-beta.

Combinatorial Game Theory

Combinatorial game theory (Conway 1976; Berlekamp, Conway, & Guy 1982) breaks up game positions into smaller pieces and analyzes the overall game in terms of these local subgames.

Each move in a game corresponds to a move in one subgame and leaves all other subgames unchanged. A game ends when all subgames have ended, and the final outcome

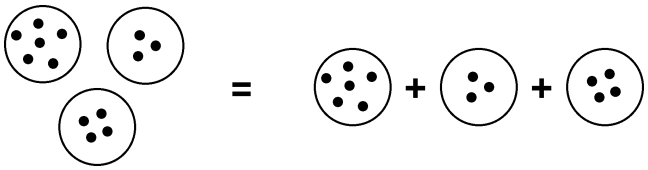


Figure 2: A three heap *Nim* position and its subgames

of the game can be determined from the subgame outcomes. A well-known example of a combinatorial game is *Nim*, shown in Figure 2, which is played with heaps of tokens. At each move, a player removes an arbitrary number of tokens from a single heap, and whoever runs out of moves first loses. Each Nim heap constitutes one subgame. While winning a single subgame is trivial, winning the *sum* of several heaps requires either exhaustive analysis, or, much more efficiently, a computation using the calculus of combinatorial games.

This theory can be seen as both a generalization and as a special case of classical minimax game theory. It is a generalization because *locally*, each player can move in each position, whereas in classical minimax games only one player has the move. On the other hand, from a global viewpoint combinatorial games are a special case, because only some games allow a decomposition into subgames.

Decomposition Search

Decomposition search (Müller 1999) is a framework for solving games through decomposition, followed by a particular kind of local search named *local combinatorial game search (LCGS)* and the analysis of the resulting local game graphs through combinatorial game theory.

Let G be a game that decomposes into a sum of subgames $G_1 + \ldots + G_n$. Let the combinatorial game evaluation of G be $C(G)$. *Decomposition search* is defined as the following four step algorithm for determining optimal play of G:

1. Game decomposition and subgame identification: given G, find an equivalent sum of subgames $G_1 + \ldots + G_n$.
2. Local combinatorial game search (LCGS): for each G_i, perform a search to find its game graph $GG(G_i)$.
3. Evaluation: for each game graph $GG(G_i)$, evaluate all terminal positions, then find the combinatorial game evaluation of all interior nodes, leading to the computation of $C(G_i)$.
4. Sum game play: through combinatorial game analysis of the set of combinatorial games $C(G_i)$, select an optimal move in $G_1 + \ldots + G_n$.

We describe steps 2 and 4 further in the following paragraphs. For further details on the theory, the algorithm, and its applications see (Berlekamp, Conway, & Guy 1982; Müller 1995; 1999).

Local Combinatorial Game Search Local combinatorial game search (LCGS) is the main information gathering step of decomposition search. It is performed independently for each subgame. LCGS generates a game graph representing all relevant move sequences that might be played locally in the course of a game. LCGS works differently from minimax tree search in a number of ways, including move generation and recognition of terminal positions.

The game graph built by LCGS also differs from the tree generated by minimax search. In the case of minimax, players move alternately, so each position is analyzed with respect to the player on move. In contrast, there is no player-to-move-next in a subgame. All possible local move sequences must be included in the analysis, including sequences with several successive moves by the same player, because players can switch between subgames at every move.

Another difference is the treatment of cycles. Classical combinatorial game evaluation is defined only for games without cycles. However, similar methods based on a technique called *thermography* are being developed that can deal with cyclic subgames as well (Berlekamp 1996; Fraser 2002; Müller 2000).

Sum Game Play

To find an optimal move in a sum game, the final step of decomposition search selects a move which most improves the position. This improvement is measured by a combinatorial game called the *incentive* of a move. The incentives of all moves in all subgames are computed locally. If one incentive dominates all others, an optimal move has been determined. This is the usual case for games with a relatively strongly ordered set of values such as Go.

Since incentives are combinatorial games and therefore only partially ordered, it can happen that more than one non-dominated candidate move remains. In this case, an optimal move is found by a more complex procedure involving the combinatorial summation of games (Conway 1976).

Since such a summation can be an expensive operation, there is no worst case guarantee that decomposition search is always more efficient than minimax search. In practice, it seems to work much better. The algorithm presents many opportunities for complexity reduction of intermediate expressions during local evaluation as well as during summation.

Even though all search and most analysis is local, decomposition search yields globally optimal play, which can switch back and forth between subgames in very subtle ways, as in the example of Figure 3.

An optimal 62 move solution sequence computed by decomposition search is shown in Figure 3. On a state of the art system 4 years ago, the complete solution took only 1.1 seconds. Full-board alpha-beta search, even with further enhancements that exploit locality, has no chance to solve this kind of problems. It requires time that is exponential in the size of the *whole problem*, whereas LCGS' worst case time is exponential in the size of the *biggest subproblem*. If the local combinatorial game evaluations generated during LCGS can be computed and compared without too much overhead, as usually seems to be the case in the Go endgames investigated, a dramatic speedup results (Müller 2001a).

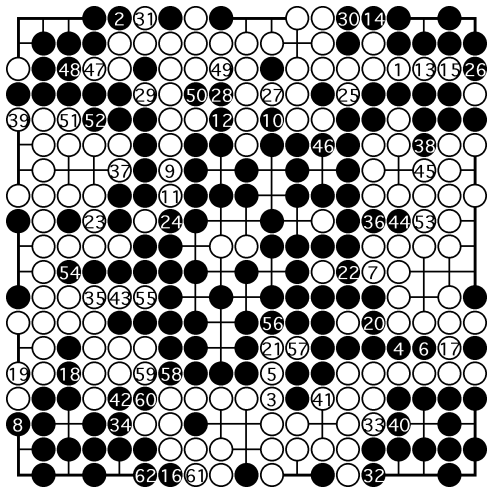


Figure 3: An optimal solution to problem C.11

Game Search Techniques in Multicriteria Planning

This section presents some preliminary ideas for applying games methods in AI planning. The author hopes that they will become part of the emerging research agenda in multicriteria planning.

Partial Order Bounding

The idea of using many simple yes/no questions to approach a complex problem is intuitively appealing. Can it be made to work in the domain of multicriteria AI planning? Many planning systems are slowed down by their manipulation of complex structures during the search. The challenge of developing a planner based on partial order bounding is to transform the planning problem into a series of decision problems that can be efficiently searched. One successful example of a similar approach are the existing methods for compiling planning problems into SAT instances.

Decomposition Search

Combinatorial game theory uses one of the fundamental approaches for dealing with complexity: divide and conquer. The unique point of this approach is the rich partially ordered structure of combinatorial games that can be used on an intermediate level, to represent solutions to subproblems. Furthermore, a powerful mathematical apparatus can be used to combine partial solutions.

An analogous approach in AI planning could work as follows:

- Split a planning problem into subtasks.
- Use local search on each subtask, which is parameterized by the possible external contexts in which it might be applied.
- Find a partial order structure to represent the parameterized solutions to subtasks.
- Define a global combination operator, which might be based on a combination of: high-level search, and knowledge about the partial order structure of sub-solutions.

To give a more concrete example, the subtask of transporting some good G from A to B might have a partially ordered solution space that is parameterized by the resources used, such as fuel, time and personnel. It can be further parameterized by the results achieved, such as the quantity of G transported, the risk of failure and so on. The idea is a "plan library" with multiattribute annotations of subtasks and solutions.

Multicriteria Planning in Games and Puzzles

Adversarial planning is more complex than single-agent planning, since normal planning usually assumes an unchanging environment under complete control of the agent, whereas in adversarial planning all possible hostile opponent actions have to be taken into account according to the minimax principle.

Go

Go is an intricate game which requires a complex evaluation (Bouzy & Cazenave 2001; Müller 2002). Most successful Go programs utilize a complex hierarchy of objects to represent the state of a Go board, and very selectively generate moves that pursue goals related to these objects. The basic evaluation in Go is a scalar measuring the balance of *territory*, often obtained by summing up a value in the range between +1 (sure point for player) and -1 (sure point for the opponent) for all points on the board. Even so, in practice many other criteria are used to modify this value (Chen 2000).

In terms of planning, high-level plans for objects on the board are used. (Hu & Lehner 1997) propose several models for combining local plans in a Go framework, taking into account the overall minimax evaluation principle as well as the question of keeping the initiative while executing one plan, which allows a player to start on the next plan as well.

Multicriteria planning appears to be a natural framework for this kind of environment. A situation can be represented as a set of plans for each player. During a game, plans are in different stages of completion, and represent different degrees of local success or failure for each player. Each move played in a game of Go typically has effects on many levels and on many different plans. Some of these effects are good for the player, while others are detrimental. This naturally leads to a partial order evaluation structure.

Planning in Sokoban

In ongoing work with Adi Botea and Jonathan Schaeffer (Botea 2002), we investigate an abstract model for planning in the puzzle of Sokoban (Junghanns 1999). This work roughly follows the decomposition search planning model outlined above. A Sokoban puzzle is split into subproblems called *rooms* and *tunnels* representing parts of the overall maze. Several static and search-based analyses are performed on the subproblems. This results in a compact intermediate representation of the possible local solutions to

each subproblem. The high-level global planner is now able to work on the much reduced abstracted planning problem instead of the original maze.

References

Allis, L. 1994. *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. Dissertation, University of Limburg, Maastricht.

Baum, E., and Smith, W. 1997. A bayesian approach to relevance in game playing. *Artificial Intelligence* 97(1-2):195–242.

Beal, D. 1999. *The Nature of Minimax Search*. Ph.D. Dissertation, Universiteit Maastricht.

Berlekamp, E.; Conway, J.; and Guy, R. 1982. *Winning Ways*. London: Academic Press. Revised version published 2001-2002 by AK Peters.

Berlekamp, E. 1996. The economist's view of combinatorial games. In Nowakowski, R., ed., *Games of No Chance: Combinatorial Games at MSRI*. Cambridge University Press. 365–405.

Botea, A. 2002. under submission.

Bouzy, B., and Cazenave, T. 2001. Computer Go: An AI-oriented survey. *Artificial Intelligence* 132(1):39–103.

Chen, K. 2000. Some practical techniques for global search in Go. *ICGA Journal* 23(2):67–74.

Conway, J. 1976. *On Numbers and Games*. London/New York: Academic Press.

Dasgupta, P.; Chakrabarti, P.; and DeSarkar, S. 1996a. Multiobjective heuristic search in AND/OR graphs. *Journal of Algorithms* 20(2):282–311.

Dasgupta, P.; Chakrabarti, P.; and DeSarkar, S. 1996b. Searching game trees under a partial order. *Artificial Intelligence* 82(1–2):237–257.

Fraser, B. 2002. *Computer-Assisted Thermographic Analysis of Go Endgames*. Ph.D. Dissertation, University of California at Berkeley. To appear.

Ginsberg, M. 2001. GIB: Imperfect information in a computationally challenging game. *JAIR* 14:303–358.

Harikumar, S., and Kumar, S. 1996. Iterative deepening multiobjective A*. *Information Processing Letters* 58(1):11–15.

Hu, S., and Lehner, P. 1997. Multipurpose strategic planning in the game of Go. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19(9):1048–1051.

Junghanns, A. 1998. Are there practical alternatives to alpha-beta? *ICCA Journal* 21(1):14–32.

Junghanns, A. 1999. *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. Dissertation, University of Alberta.

Lincke, T. 2002. *?* Ph.D. Dissertation, ETH Zurich. To appear.

Müller, M. 1995. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. Ph.D. Dissertation, ETH Zürich. Diss. ETH Nr. 11.006.

Müller, M. 1999. Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames. In *IJCAI-99*, 578–583.

Müller, M. 2000. Generalized thermography: A new approach to evaluation in computer Go. In van den Herik, J., and Iida, H., eds., *Games in AI Research*, 203–219. Maastricht: Universiteit Maastricht.

Müller, M. 2001a. Global and local game tree search. *Information Sciences* 135(3–4):187–206.

Müller, M. 2001b. Partial order bounding: A new approach to evaluation in game tree search. *Artificial Intelligence* 129(1-2):279–311.

Müller, M. 2002. Computer Go. *Artificial Intelligence* 134(1–2):145–179.

Ore, O. 1962. *Theory of Graphs*, volume 38 of *Colloquium Publications*. Providence: American Mathematical Society.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley Publishing Company: Reading, Massachusetts. ISBN 0-201-05594-5.

Plaat, A.; Schaeffer, J.; Pijls, W.; and De Bruin, A. 1996. Best-first fixed-depth minimax algorithms. *Artificial Intelligence* 87:255–293.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: a Modern Approach*. Prentice Hall.

Stanley, R. 1997. *Enumerative Combinatorics Vol. 1*. Number 49 in Cambridge Studies in Advanced Mathematics. Cambridge University Press.

Stewart, B., and White, C. 1991. Multiobjective A*. *Journal of the ACM* 38(4):775–814.

Trotter, W. 1992. *Combinatorics and Partially Ordered Sets. Dimension Theory*. Baltimore: Johns Hopkins University Press.

von Neumann, J., and Morgenstern, O. 1947. *Theory of Games and Economic Behaviour*. Princeton University Press, Princeton, 2nd edition.

von Neumann, J. 1928. Zur Theorie der Gesellschaftsspiele. *Math. Ann.* 100:295–320.

Yanez, J., and Montero, J. 1999. A poset dimension algorithm. *Journal of Algorithms* 30:185–208.

Algorithms for Routing with Multiple Constraints

Anuj Puri[*] and **Stavros Tripakis**[†]

Abstract

In this paper, we study the problem of routing under multiple constraints. We consider a graph where each edge is labeled with a cost and a delay. We then consider the problem of finding a path from a source vertex to a destination vertex such that the sum of the costs on the path satisfy the cost constraint and the sum of the delays satisfy the delay constraint. We present three different algorithms for solving the problem. These algorithms have varying levels of complexity and solve the problem with varying degrees of accuracy. We present an implementation of these algorithms and discuss their performance on different graphs.

Introduction

Finding paths in a graph with respect to multiple criteria is a fundamental problem, with applications in many areas. For example, one such application might be finding routes in a communication network, with respect to different quality-of-service criteria (delay, cost, packet loss, etc). Another application is to find the shortest-delay route in a given map, while avoiding critical (say, unsafe) regions in the map.

In this paper, we reconsider the problem of finding paths satisfying multiple constraints. We simplify our presentation by considering only two constraints (which we call *delay* and *cost*, for simplicity), and we discuss at the end of the paper extensions to more than two constraints. Our objective is to find a path from a given source node in a graph to a given destination node, such that the sum of all delays on the path is less than a given D, and the sum of all costs is less than a given C. Solving this problem exactly is well known to be NP-Complete (Garey and Johnson 1979), (Jaffe 1984), (Hassin 1992).

We present three different algorithms for solving the problem. The first algorithm is a pseudo-polynomial time algorithm which solves the problem exactly in time $O(|V||E|min\{C, D\})$ where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The algorithm either reports back with a path satisfying the constraints or states that no such path exists. The second algorithm solves the problem approximately but with an error of at most ϵ. That is, either it states that no path satisfying the constraints exists, or it finds a path such that the sum of costs on the path is at most $C \cdot (1 + \epsilon)$, and the sum of delays is at most $D \cdot (1 + \epsilon)$. The complexity of this algorithm is $O(|V|^2|E|(1+\frac{1}{\epsilon}))$. The third algorithm finds a path with an error of at most $\epsilon = 1$. This algorithm requires a solution of the shortest path problem on the given graph. Although most of the paper is focused on dealing with two constraints, the first two algorithms generalize in a straightforward manner to more than two constraints.

Relationship to other work The routing problem with more than one constraint has been studied by several researchers. It is well known that the problem is NP-Complete (Garey and Johnson 1979), (Jaffe 1984), (Hassin 1992). An explicit proof of this is provided in (Wang and Crowcroft 1996). In (Jaffe 1984), a pseudo-polynomial time algorithm is presented for exactly solving the problem with complexity $O(|V|^5 max\{C, D\}log(|V|max\{C, D\}))$. By a more careful analysis and using the data structures in a more clever manner, we show that the complexity of our algorithm (which is similar to the one of (Jaffe 1984)) is $O(|V||E|min\{C, D\})$. In (Jaffe 1984), an approximation algorithm that solves the problem with approximation error $\epsilon = 1$ using the shortest path algorithm is also presented. Although this is similar to our shortest-path based algorithm, our algorithm in general will perform better because we solve a series of shortest path problems, each obtaining a better solution than the last one. In (Hassin 1992), several algorithms are presented for approximating the solution to the problem for acyclic graphs. The complexity of the two approximation algorithms are $O(loglogB(\frac{|V||E|}{\epsilon} + loglogB))$ and $O(|E|\frac{|V|^2}{\epsilon}log(\frac{n}{\epsilon}))$ where ϵ is the error of the approximation and $B = max\{C, D\}$. Our approximation algorithm works for all graphs and has complexity $O(|V|^2|E|(1 + \frac{1}{\epsilon})$. The algorithms also use somewhat different techniques. Our algorithm is essentially a generalization of the Bellman-Ford algorithm where we keep track of errors during the iteration.

The multi-constrained cost-delay routing problem is also considered in (Salama et al. 1997), (Chen and Nahrstedt 1998a), (Chen and Nahrstedt 1998b), (Orda 1999). In (Chen and Nahrstedt 1998a), the authors propose an algorithm pa-

[*]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720

[†]Verimag, Centre Equation, 2, avenue de Vignate, 38610, Gières, France

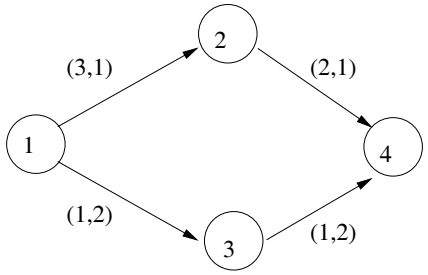


Figure 1: A Simple Network

rameterized by an integer x, where if a solution is found then that solution is feasible in the original problem. On the other hand a solution might not be found even if it exists. The complexity of the algorithm is $O(xVE)$. The larger x, the higher the chances to find a solution if it exists, but it is not clear how to choose x, although some sufficient conditions are provided for x. In (Salama et al. 1997), a distributed heuristic algorithm is proposed with complexity $O(|V|^2)$. No bound on the error of the algorithm is given, but simulations are provided which show that error is within 10% of the optimal. In (Orda 1999), an ϵ-approximative algorithm is proposed for a particular class of hierarchical networks where the topology must satisfy a number of conditions. A comprehensive review of work on different QOS unicast and multicast routing problems, including multi-constrained routing is given in (Chen and Nahrstedt 1998b).

Problem Formulation

We consider a directed *2-weight* graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. An edge $e \in E$ is $e = (v, w, c, d)$ where the edge goes from v to w, and has delay $\mathsf{delay}(e) = d$ and $\mathsf{cost}(e) = c$. We write this as $v \stackrel{(c,d)}{\longrightarrow} w$. When there is no confusion, we may also write the edge as (v, w) and say the edge is labeled with (c, d).

A path is $p = v_1 \stackrel{(c_1,d_1)}{\longrightarrow} v_2 \stackrel{(c_2,d_2)}{\longrightarrow} v_3 \stackrel{(c_3,d_3)}{\longrightarrow} \cdots \stackrel{(c_n,d_n)}{\longrightarrow} v_{n+1}$. The cost of a path is $\mathsf{cost}(p) = \sum_{i=1}^{n} c_i$ and its delay is $\mathsf{delay}(p) = \sum_{i=1}^{n} d_i$.

Given a path p and cost constraint $C \geq 1$ and delay constraint $D \geq 1$, we say p is *feasible* provided $\mathsf{cost}(p) \leq C$ and $\mathsf{delay}(p) \leq D$. The problem of *routing under two constraints* is, given $G = (V, E)$, cost constraint C and delay constraint D, a *source* node $s \in V$ and a *destination* node $t \in V$, find a feasible path p from s to t, or decide that no such path exists.

Example 1 *Consider the 2-weight graph of Figure 1. Each edge is labeled with (c, d) where c is the cost of the edge and d is the delay of the edge. For example, the edge from vertex 1 to vertex 2 has cost 3 and delay 1. Suppose the source vertex is 1, the destination vertex is 4, the cost constraint is $C = 5$ and the delay constraint is $D = 2$. Then, the path $1 \stackrel{(3,1)}{\longrightarrow} 2 \stackrel{(2,1)}{\longrightarrow} 4$ is feasible, whereas $1 \stackrel{(1,2)}{\longrightarrow} 3 \stackrel{(1,2)}{\longrightarrow} 4$ is not (since it violates the delay constraint).*

The reader can check that if $C = 4$ and $D = 3$, then there is no feasible path.

Rather than checking to see if a graph has a feasible path, it is sometimes useful to try to minimize the following objective function

$$M(p) = max\{\frac{max\{\mathsf{cost}(p), C\}}{C}, \frac{max\{\mathsf{delay}(p), D\}}{D}\}.$$

Observe that for any path p, $M(p) \geq 1$ and $M(p) = 1$ iff p is feasible. But even if a feasible path does not exist or is hard to find, by trying to minimizing $M(p)$ we can get a path that comes "close" to satisfying the constraints.

Formally, we define the error of a path p as

$$\mathsf{error}(p) = \frac{M(p) - M(p^*)}{M(p^*)}$$

where p^* is the path which minimizes M (in case more than one paths minimize M, we pick p^* arbitrarily among them, since the minimal value $M(p^*)$ is the same for all of them).

Notice that $\mathsf{error}(p) \geq 0$ and $\mathsf{error}(p) = 0$ iff p is feasible. Also note that if $\mathsf{cost}(p) \leq C \cdot (1 + \epsilon)$ and $\mathsf{delay}(p) \leq D \cdot (1 + \epsilon)$ then $\mathsf{error}(p) \leq \epsilon$. Indeed, the two above conditions imply that $M(p) \leq 1 + \epsilon$ and, since $M(p^*) \geq 1$, we get $\mathsf{error}(p) \leq \epsilon$.

In case it is too difficult to find p^*, we look for a path p for which $\mathsf{error}(p)$ is small. We next present algorithms for finding a feasible path and for finding paths for which $\mathsf{error}(p)$ is small.

Complexity

The multiple-constraint routing problem is known to be NP-complete. For completeness of the paper, we provide a proof here as well.

Theorem 1 *The routing problem with two constraints is NP-Complete.*

Proof: We will provide a reduction from the knapsack problem. Recall that in the knapsack problem, we are given positive integers $c_1, c_2, \ldots, c_n$, and N, and the objective is to find a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} c_i = N$.

From the knapsack problem, we construct a graph with vertices $\{1, \ldots, n\}$. There are two edges from vertex i to vertex $i + 1$: edge $(i, i + 1, c_i, 0)$ and edge $(i, i + 1, 0, c_i)$. Figure 2 shows the scenario. Our objective is to find a path from vertex 1 to vertex n with cost constraint N and delay constraint $\sum_{i=1}^{n} c_i - N$. It is easy to check that there is a path that satisfies the constraints iff there is a solution to the knapsack problem. ∎

A pseudo-polynomial algorithm

In this section, we propose an algorithm for the problem of routing under two constraints with worst-case complexity $O(|V| \cdot |E| \cdot min\{C, D\})$. That is, the algorithm is polynomial on the size of the graph (quadratic on the number of vertices and linear on the number of edges), but also linearly depends on the smaller of the bounds C and D. Therefore, it is a *pseudo*-polynomial algorithm.

Let us begin by making a safe hypothesis. Given a 2-weight graph $G = (V, E)$, where $|V| = n$, let $\mathsf{cost}_{max} = max\{c \mid (\_, \_, c, \_) \in E\}$ and $\mathsf{delay}_{max} = max\{d \mid$

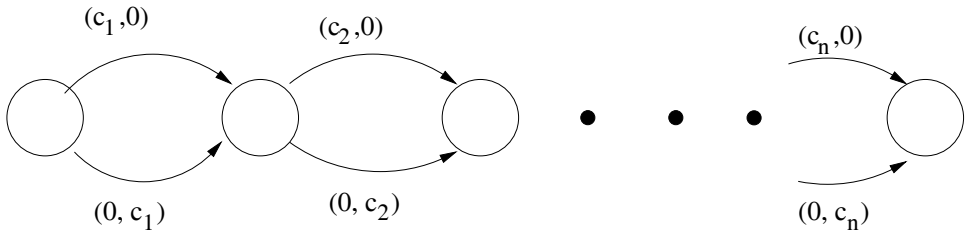


Figure 2: Graph obtained from the knapsack problem

$(\_, \_, d, \_) \in E\}$ be the maximum cost and delay associated with any edge of G. Now, assume that $n \cdot \mathsf{cost}_{max} \leq C$. Then, given $u, v \in V$, there exists a feasible path from u to v iff there exists a path p from u to v such that $\mathsf{delay}(p) \leq D$. Given this observation, finding a feasible path in G from u to v comes down to finding the smallest-delay path from u to v, that is, the path p that minimizes $\mathsf{delay}(p)$. This can be easily done using a shortest-path algorithm, with cost $O(|V| \cdot |E|)$. Since this is less than $O(|V| \cdot |E| \cdot min\{C, D\})$, this case is not interesting. The case where $n \cdot \mathsf{delay}_{max} \leq D$ is symmetric.

So, from now on we assume that $n \cdot \mathsf{cost}_{max} > C$ and $n \cdot \mathsf{delay}_{max} > D$. We also assume that the greatest common divisor of $\{C, \mathsf{cost}(e) \mid e \in E\}$ is 1, and similarly for the delays (otherwise we could just divide all costs/delays by their greatest common divisor, without affecting the problem).

Informally, our algorithm works as follows. For each vertex w, we compute a set of *cost-delay pairs* F_w. Each $(c, d) \in F_w$ will represent the cost and delay of a possible path from w to the destination vertex v. To keep the size of F_w manageable, we eliminate from F_w all elements corresponding to infeasible paths (i.e., all (c, d) such that $c > C$ or $d > D$). Moreover, we eliminate from F_w all redundant elements, that is, all elements with both cost and delay greater from some other element. Let us make these more precise below.

Cost-delay sets

A *cost-delay set* for a vertex w is a set $F_w \subseteq \mathsf{N} \times \mathsf{N}$. An element (c, d) of F_w is called *infeasible* if either $c > C$ or $d > D$. An element (c, d) of F_w is called *redundant* if there exists a different $(c', d') \in F_w$ such that $c' \leq c$ and $d' \leq d$.

A cost-delay set F is said to be *minimal* if it contains no infeasible or redundant elements. The following properties hold (assuming C and D fixed):

Proposition 1 *If F is minimal, then $|F| \leq min\{C, D\}$. To every cost-delay set F corresponds a unique greatest minimal subset $F' \subseteq F$.*

We write $\mathsf{minim}(F)$ to denote the greatest minimal subset of F.

Figure 3 displays the typical structure of a cost-delay set and its minimal. Black and grey bullets are infeasible and redundant elements, respectively.

Minimal cost-delay sets admit an efficient canonical representation as sorted lists. Consider a minimal set $F = \{(c_1, d_1), (c_2, d_2), ..., (c_n, d_n)\}$ and assume, without loss of

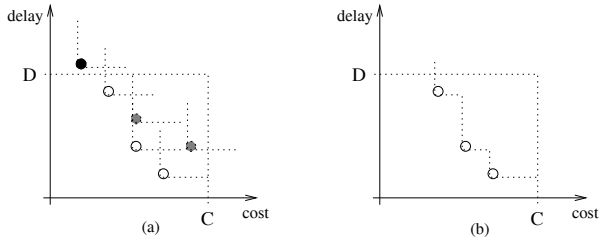


Figure 3: A cost-delay set (a) and its minimal (b)

generality, that $c_1 \leq c_2 \leq \cdots \leq c_n$. Then, $d_1 \geq d_2 \geq \cdots \geq d_n$ must hold, otherwise there would be at least one redundant element in F. Consequently, F can be represented as the list $(c_1, d_1)\ (c_2, d_2)\ \cdots\ (c_n, d_n)$, sorted using cost as the "key". This representation is canonical in the sense that two minimal sets F_1, F_2 are equal iff their list representations are identical.

The algorithm works with minimal cost delay sets and uses two operations, namely, union and translation with respect to a vector $(c, d) \in \mathsf{N}^2$. We present these operations and discuss how they can be implemented using sorted-lists and preserving the canonical representation.

Given minimal (i.e., feasible and non-redundant) F_1, F_2, the union $F_1 \cup F_2$ is always feasible, but not necessarily non-redundant. In order to compute $F = \mathsf{minim}(F_1 \cup F_2)$ directly from the list representations L_1, L_2 of F_1, F_2, we can use a simple modification of a usual merge-sort algorithm on lists. The latter takes as input L_1, L_2 and produces L, the list representation of F. In order to guarantee the absence of redundant points in L, it compares at each step the heads (c_1, d_1) and (c_2, d_2) of (the remaining parts of) L_1, L_2. If $c_1 \leq c_2$ and $d_1 \leq d_2$ then (c_2, d_2) is redundant and is skipped. If $c_2 \leq c_1$ and $d_2 \leq d_1$ then (c_1, d_1) is skipped. Otherwise, the pair with the smallest c_i is inserted in L and the head pointer move one element ahead in the corresponding list L_i. It is easy to see that this algorithm is correct. The cost of the algorithm is $n_1 + n_2$, where n_i is the length of L_i. Therefore, from Proposition 1, the worst-case complexity of computing the union of cost-delay sets is $O(min\{C, D\})$.

Translation is defined on a cost-delay set F and a pair $(c, d) \in \mathsf{N}^2$:

$$F + (c, d) \stackrel{\text{def}}{=} \{(c' + c, d' + d) | (c', d') \in F\}$$

If F is minimal, then $F + (c, d)$ is non-redundant, how-

ever, it may contain infeasible points. These can be easily eliminated, however, while building the list L' for $min(F + (c,d))$: the list of F is traversed, adding (c,d) to each of its elements, (c_i,d_i); if $c_i + c \leq D$ and $d_i + d \leq D$ then $(c_i + c, d_i + d)$ is inserted at the end of L', otherwise it is infeasible and it is skipped. At the end, L' will be sorted by cost. The complexity of translation is $O(min\{C,D\})$.

The algorithm

The algorithm iteratively computes the (minimal) cost-delay sets of all vertices in the graph. Let F_w^j denote the cost-delay set for vertex w at iteration j. Initially, all vertices have empty cost-delay sets, $F_w^0 = \emptyset$, except v, for which $F_v^0 = \{(0,0)\}$. At each iteration, each vertex updates its cost-delay set with respect to all its successor vertices. Computation stops when no cost-delay set is updated any more. We now present the operations performed at each iteration at each vertex w.

Let $w_1, ..., w_k$ be the successor vertices of w, that is, $w \xrightarrow{(c_i,d_i)} w_i$, for $i = 1, ..., k$ (note that $w_1, ..., w_k$ might not be distinct). Then, the cost-delay set of w at iteration $j+1$ will be:

$$F_w^{j+1} = \mathsf{minim}\Big(F_w^j \cup \bigcup_{i=1}^{k} \big(F_{w_i}^j + (c_i,d_i)\big)\Big) \qquad (1)$$

That is, we add to the possible cost-delay values for w all values obtained by taking an edge to some successor vertex w_i, and then continuing with a possible cost-delay value for w_i.

The following proposition proves termination and correctness of the algorithm.

Proposition 2 *(Termination) The updating of the cost-delay sets will stabilize after at most $|V|$ iterations, that is, for any vertex w, $F_w^{|V|+1} = F_w^{|V|}$. (Correctness) A feasible path from w to v exists iff $F_w^{|V|} \neq \emptyset$. For any $(c,d) \in F_w^{|V|}$, there exists a path p from w to v such that $\mathsf{cost}(p) = c$ and $\mathsf{delay}(p) = d$.*

Worst-case complexity of the algorithm

Proposition 2 implies that the algorithm stops after at most $|V|$ iterations. At each iteration, the cost-delay set of each vertex is updated with respect to all its successor vertices. Thus, there are at most $|E|$ updates at each iteration. Each update involves a translation and a union, both of which have complexity $O(min\{C,D\})$. Therefore, the overall worst-case complexity of the algorithm is $O(|V| \cdot |E| \cdot min\{C,D\})$.

Incorporating routing information

As defined, cost-delay sets do not contain any routing information, that is, at the end of the algorithm, we know that a point in F_w represents the cost-delay value of a possible feasible path from w to v, but we do not know which path. This information is easy to incorporate, at the expense of associating to each $(c,d) \in F_w$, the edge $e = (w, w_1, c', d')$, and a pointer to the element $(c_1,d_1) \in F_{w_1}$, from which (c,d) was generated. The edge and (c_1,d_1) element are unique, and come from the operation $F_w \cup (F_{w_1} + (c',d'))$. In order to reconstruct the path from w with cost-delay (c,d) we follow the edge e to w_1, then look for the path from w_1 with cost-delay (c_1,d_1), and so on.

A bounded-error approximative algorithm

In this section we give an approximative algorithm for the problem of routing under two constraints. The algorithm is approximative in the sense that, it might not yield a feasible path, even if such a path exists. However, the error in the path p returned by the algorithm can be bounded: $\mathsf{error}(p) \leq \epsilon$, where ϵ is an input parameter. The algorithm has worst-case complexity $O(|V|^2 \cdot |E| \cdot (1 + \frac{1}{\epsilon}))$, which implies that it is worth using only when $|V|$ is (much) smaller than $\frac{\epsilon}{1+\epsilon} \cdot min\{C,D\}$. Otherwise, the pseudo-polynomial algorithm, being exact and less expensive, would be preferable. In the rest of this section we assume that $|V| < \frac{\epsilon}{1+\epsilon} \cdot min\{C,D\}$.

Minimal-distance cost-delay sets

The approximative algorithm is similar to the pseudo-polynomial algorithm, with the additional fact that it eliminates elements of cost-delay sets which are "too close" to some other element. More formally, for $(c_1,d_1),(c_2,d_2) \in \mathsf{N}^2$, define:

$$||(c_1,d_1),(c_2,d_2)|| \stackrel{\text{def}}{=} max\{|c_1 - c_2|, |d_1 - d_2|\}$$

Then, a cost-delay set F is said to have *minimal distance* δ iff for all distinct $(c_1,d_1),(c_2,d_2) \in F$, $||(c_1,d_1),(c_2,d_2)|| \geq \delta$.

Given a cost-delay set F and some $\delta \geq 2$, we would like to find a subset $F' \subseteq F$, such that:

1. F' has minimal distance δ, and
2. for all $x \in F - F'$, there exists $y \in F'$ such that $||x,y|| < \delta$.

Condition 2 ensures that no elements of F are dropped unnecessarily (were condition 2 to be missed, the trivial subset $F' = \emptyset$ would satisfy condition 1). A subset $F' \subseteq F$ satisfying the above conditions is called a *maximal δ-distance subset of F*. In general, there may be more than one maximal δ-distance subsets of a given F (any one of them is good for our purposes). We now give a procedure to compute, given F, a maximal δ-distance subset $F' \subseteq F$.

The procedure takes as input the list representation L of F and generates as output a list L'. Assume $L = (x_1, ..., x_n)$. Initially, $L' = (x_1)$. Let y denote the last element of L', at each point during the execution of the procedure. For each $i \geq 2$, if $||x_i, y|| \geq \delta$ then x_i is appended at the end of L' and y is updated to x_i, otherwise, x_i is skipped. It can be shown that the list built that way represents a legal δ-distance subset of F. From now on, we denote this set by $\mathsf{min\_dist}(\delta, F)$.

Definition 1 *We define the* step error, *δ_ϵ, to be $\frac{min\{C,D\} \cdot \epsilon}{|V|}$.*

The algorithm

The approximative algorithm is obtained from the pseudo-polynomial algorithm with the following modification. Given $\epsilon \in [0,1]$, instead of keeping a minimal set F_w for each node w, we keep a set B_w such that:

1. B_w has no redundant elements,
2. for each $(c,d) \in B_w$, $c \leq (1+\epsilon)\cdot C$, $d \leq (1+\epsilon)\cdot D$ (that is, the feasibility region is extended by $(\epsilon \cdot C, \epsilon \cdot D)$,
3. B_w has minimal distance δ_ϵ.

That is, in the approximative algorithm, the fix-point equations are as follows:

$$B_w^{j+1} = \mathsf{min\_dist}\Big(\delta_\epsilon, \mathsf{minim}\Big(B_w^j \cup \bigcup_{i=1}^n \big(B_{w_i}^j + (c_i,d_i)\big)\Big)\Big)$$

As in the case of the exact algorithm, termination of the approximative algorithm is ensured in $|V|$ steps.

Proposition 3 *Consider a graph G, nodes u, v of G, and cost-delay constraints C, D. Then, for given ϵ:*

(1) If $B_u = \emptyset$ at the end of the approximative algorithm, then no feasible path from u to v exists.

(2) If $B_u \neq \emptyset$, then for each $(c,d) \in B_u$, there exists a path p from u to v such that $\mathsf{cost}(p) = c$, $\mathsf{delay}(p) = d$ and $\mathsf{error}(p) \leq \epsilon$.

Proof (sketch):

Let w be a node and F_w, B_w be the final cost-delay sets computed for w by the exact and approximative algorithms, respectively. The result is based on the fact that, for any $(c,d) \in F_w$, there exists $(c',d') \in B_w$, such that $||(c,d),(c',d')|| \leq |V|\cdot\delta_\epsilon$. This is because at most δ_ϵ "error" accumulates at each step of the algorithm, when eliminating pairs during the $\mathsf{min\_dist}$ operation.

By definition of δ_ϵ, we have that $||(c,d),(c',d')|| \leq min\{C,D\}\cdot\epsilon$. Then, assuming (c,d) to be the cost and delay of an optimal path p^* and (c',d') the cost and delay of a path p computed by the approximative algorithm, it is easy to prove that $\mathsf{error}(p) \leq \epsilon$. For (1), notice that if p^* is feasible then $c' \leq (1+\epsilon)\cdot C$ and $d' \leq (1+\epsilon)\cdot D$, This means that (c',d') is indeed "inside" the extended feasibility region, thus, is not eliminated from B_w during the approximative algorithm. ∎

Worst-case complexity

The only difference from the pseudo-polynomial algorithm is in the worst-case size of the cost-delay sets B_w. Since the latter have minimal distance δ_ϵ and are bounded by the feasibility region $((1+\epsilon)\cdot C, (1+\epsilon)\cdot D)$, we have $|B_w| \leq \frac{(1+\epsilon)\cdot min\{C,D\}}{\delta_\epsilon}$. By definition of δ_ϵ, we get $|B_w| \leq \frac{(1+\epsilon)}{\epsilon}|V|$. The union, translation and $\mathsf{min\_dist}$ operations can be implemented using sorted lists to represent the sets B_w (the canonical representation is not affected by minimal distance). The cost of the operations is, as previously, linear on the size of the lists, which yields an overall worst-case complexity of $O(|V|^2\cdot|E|\cdot(1+\frac{1}{\epsilon}))$.

Satisfying constraints by using the shortest path algorithm

In this section, we consider an algorithm for finding a path which satisfies the two constraints by using the shortest path algorithm. Our objective will be to use the shortest path algorithm to find a path p which minimizes $M(p)$.

For the rest of this section we assume that we have normalized the costs and delays by dividing the costs by C and the delays by D.

A path is feasible in the new graph if $\mathsf{cost}(p) \leq 1$ and $\mathsf{delay}(p) \leq 1$. Note that a path is feasible in the new graph iff it was feasible in the original graph. Furthermore, $M(p)$ is the same in both graphs.

To find a path satisfying two constraints by using the shortest path algorithm, we choose an $0 \leq \alpha \leq 1$ and replace the cost c and the delay d associated with an edge with the weight $\alpha c + (1-\alpha)d$. We then use the shortest path algorithm to find a path with the smallest weight. We refer to this path as $SP(G,\alpha)$. As the next lemma shows, $p = SP(G,\alpha)$ has an error $\mathsf{error}(p)$ of at most 1 for $\alpha = \frac{1}{2}$.

Lemma 1 *For a graph $G = (V,E)$, $M(p^*) \leq M(p) \leq 2M(p^*)$, where $p = SP(G,\alpha)$ and p^* is the path which minimizes M.*

Proof: Recall that for all paths p', $M(p') \geq 1$. If $M(p) = 1$, then clearly $M(p) \leq 2M(p^*)$. So assume $M(p) > 1$. Then $M(p) \leq \mathsf{cost}(p) + \mathsf{delay}(p) \leq \mathsf{cost}(p^*) + \mathsf{delay}(p^*) \leq 1 + 1 \leq M(p^*) + M(p^*) = 2M(p^*)$. ∎

The previous lemma shows that by choosing $\alpha = \frac{1}{2}$, we can obtain a path p with $\mathsf{error}(p) \leq 1$. We now present an algorithm that minimizes $M(SP(G,\alpha))$ by choosing the appropriate α.

The algorithm uses binary search: assume we know that the optimal value of α lies in the interval $[l,u]$; we find $p = SP(G,\alpha)$ for $\alpha = \frac{l+u}{2}$; if $\mathsf{cost}(p) \leq \mathsf{delay}(p)$, we eliminate the interval $(\frac{l+u}{2}, u]$ from consideration, otherwise, we eliminate $[l, \frac{l+u}{2})$. The algorithm terminates when $SP(G,l) = SP(G,u)$.

The reason that half of the interval can be eliminated follows from the following lemma.

Lemma 2 *Suppose $p = SP(G,\alpha)$ and $\mathsf{cost}(p) \leq \mathsf{delay}(p)$. Then for $\alpha' \geq \alpha$ and $p' = SP(G,\alpha')$, $\mathsf{cost}(p') \leq \mathsf{cost}(p)$ and $\mathsf{delay}(p') \geq \mathsf{delay}(p)$.*

Proof: There are four cases:

1. $\mathsf{cost}(p') > \mathsf{cost}(p)$ and $\mathsf{delay}(p') > \mathsf{delay}(p)$.
2. $\mathsf{cost}(p') < \mathsf{cost}(p)$ and $\mathsf{delay}(p') < \mathsf{delay}(p)$.
3. $\mathsf{cost}(p') > \mathsf{cost}(p)$ and $\mathsf{delay}(p') < \mathsf{delay}(p)$.
4. $\mathsf{cost}(p') \leq \mathsf{cost}(p)$ and $\mathsf{delay}(p') \geq \mathsf{delay}(p)$.

Case 1 is not feasible because then path p improves on $p' = SP(G,\alpha')$. Case 2 is not feasible because then path p' improves on $p = SP(G,\alpha)$. Case 3 is not feasible because $\alpha\mathsf{cost}(p') + (1-\alpha)\mathsf{delay}(p') \geq \alpha\mathsf{cost}(p) + (1-\alpha)\mathsf{delay}(p)$ and $(\alpha'-\alpha)\mathsf{cost}(p') + (\alpha-\alpha')\mathsf{delay}(p') > (\alpha'-\alpha)\mathsf{cost}(p) + (\alpha-\alpha')\mathsf{delay}(p)$, and hence $\alpha'\mathsf{cost}(p') + (1-\alpha')\mathsf{delay}(p') > \alpha'\mathsf{cost}(p) + (1-\alpha')\mathsf{delay}(p)$ — a contradiction since $p' = SP(G,\alpha')$. Therefore, 4 is the only feasible case. ∎

Now assume we found $p = SP(G,\alpha)$ and $\mathsf{delay}(p) > 1$ and $\mathsf{cost}(p) \leq \mathsf{delay}(p)$. Then from Lemma 2, for $\alpha' \geq \alpha$, for $p' = SP(G,\alpha')$, $\mathsf{cost}(p') \leq \mathsf{cost}(p)$ and $\mathsf{delay}(p') \geq \mathsf{delay}(p)$. Therefore $M(p') \geq M(p)$, and hence the interval $(\alpha, u]$ can be eliminated from consideration. By similar reasoning, if $\mathsf{delay}(p) \leq \mathsf{cost}(p)$, then the interval $[l,\alpha)$ can be eliminated.

Here is a more formal statement of the algorithm:

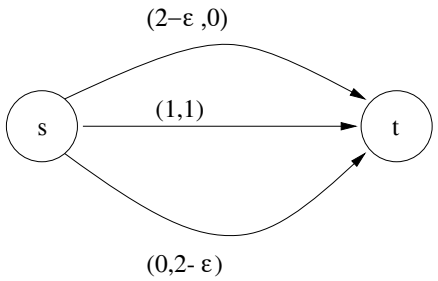


Figure 4: A graph for which the error is $\mathsf{error}(SP(G,\alpha)) = 1-\epsilon$ for all $0 \le \alpha \le 1$

Algorithm to find α to minimize $M(SP(G,\alpha))$:

```
l = 0
u = 1
p_l = SP(G, l)
p_u = SP(G, u)
Repeat
   α = (l+u)/2
   p_α = SP(G, α)
   if ( cost(p) ≤ delay(p) )
      l = α
      p_l = p_α
   else
      u = α
      p_u = p_α
Until (p_l = p_u)
```

Theorem 2 *The above algorithm terminates in polynomial number of steps, and the α^* computed by the algorithm satisfies $M(SP(G,\alpha^*)) \le M(SP(G,\alpha))$ for $\alpha \in [0,1]$.*

Notice that, although $\mathsf{error}(SP(G,\alpha^*)) \le 1$ (Lemma 1), there are "bad" examples where the error can be arbitrarily close to 1.

Example 2 *Consider the example in Figure 4 where the cost constraint is $C = 1$ and the delay constraint is $D = 1$. It is easy to check that for any $0 \le \alpha \le 1$, $\mathsf{error}(SP(G,\alpha)) = 1-\epsilon$.*

A Linear Programming solution to a relaxed problem

In this section, we relax the requirements of our problem. Rather than asking for a single path that satisfies the cost and delay requirements, we allow for the data to be routed over multiple paths. But we require the average delay and average cost to satisfy the constraints.

Let us define f_e to be fraction of the data from the source to the destination that flows over the edge e. We then have the following constraints ($out(v)$ are the outgoing edges and $in(v)$ are the incoming edges of a node v):

$$\text{For each} \quad e \in E, f_e \ge 0$$

$$\sum_{e \in out(s)} f_e = 1 \tag{2}$$

$$\sum_{e \in in(t)} f_e = 1 \tag{3}$$

$$\text{For} \quad v \ne s \ \text{and} \ \ v \ne t, \ \sum_{e \in in(v)} f_e = \sum_{e \in out(v)} f_e \tag{4}$$

$$\sum_{e \in E} f_e \ \mathsf{cost}(e) \le C \tag{5}$$

$$\sum_{e \in E} f_e \ \mathsf{delay}(e) \le D \tag{6}$$

Equation 2- 4 are the balance equations for the nodes. Equation 5 states that the average cost must be less than the cost constraint C, and Equation 6 states that the average delay must be less than the delay constraint D.

A feasible solution of the above linear program tells us how the data should be routed from the source to the destination so that average cost and delay constraints are satisfied.

Example 3 *Consider again Example 1 and Figure 1 with cost constraint 4 and delay constraint 3. If we formulate the above set of linear constraints for this problem, we note that $f_e = \frac{1}{2}$ for each edge e is a solution. This means that half of the data from the source is routed to node 2, and the other half to node 3. The average delay corresponding to this solution is $\frac{1}{2}(2+2+1+1) = 3$, and the average cost is $\frac{1}{2}(3+2+1+1) = 3.5$. Notice that even though the average cost and delay satisfy the constraints, individual paths may not (e.g., the path $(1,3)(3,4)$ does not satisfy the delay constraint).*

If we restrict ourselves to integer solutions of the above linear program (i.e, a integer programming problem), then each solution represents a single path that satisfies the delay and cost constraints. Of course, checking feasibility of integer linear programs is NP-complete.

Other Extensions

In this section we discuss the extension of the problem to the case with more than two constraints. We also discuss a somewhat different, but sometimes more useful problem in practice, where we minimize the cost subject to a delay constraint.

More than two constraints

In a problem with k constraints, we are given a k-weight graph, where each edge is labeled with a k-tuple $(c_1, c_2, \ldots, c_k)$. We are required to find a path such that the sum of the ith weight along the path is less than a bound C_i.

By a straightforward extension of the pseudo-polynomial and bounded-error algorithms, it is easy to show that we can get an exact algorithm with complexity $O(|V||E|\prod_{i=1}^{k} C_i)$, and a bounded-error approximative algorithm with complexity $O(|V|^k|E|(1+\frac{1}{\epsilon})^k)$, where ϵ is the maximum error allowed. The basic idea of the extension is that cost-delay sets now become general *Pareto* sets containing k-tuples of the form $(a_1, \ldots, a_k)$. Such a tuple in the Pareto set associated with some vertex w means that it is possible to get from w to

the destination vertex along a path in which the sum of the ith weight is a_i.

It is also possible to extend some parts of the shortest-path based algorithm It is possible to obtain a path with error $\epsilon = k - 1$ for a problem with k constraints by solving the shortest path algorithm. But it is not clear how to extend the algorithm which iterates over shortest path problems to the case with more than two constraints.

An alternative formulation

An alternative and sometimes more useful formulation is when a bound is given on the delay, and subject to this, we are required to minimize the cost. The pseudo-polynomial and bounded-error algorithms can be straightforwardly extended to solve this problem. In the final cost-delay set of the source node, we find the pair (c_i, d_i) with the smallest c_i. This corresponds to an optimal path to the destination with minimal cost c_i.

To be able to solve this alternate formulation, we also augmented the shortest-path based algorithm. To find a path which meets the delay constraint D and has minimum cost C, we solve a problem with delay constraint D and cost constraint C where C is initially chosen to be large. We then find the minimum cost by performing a binary search on C.

In the experimental results presented below, this alternative formulation of the problem is solved.

Experimental results

We have implemented in C the bounded-error approximative algorithm and the shortest-path based algorithm. In this section, we report results obtained by applying the algorithms on a number of multi-weight graphs. Our objective was to see how well the algorithms perform on graphs of medium to large size. Also, to check how sensitive the algorithms were to different parameters (e.g., number of weights, source/destination pairs, step-error).

The graphs were obtained by translating elevation maps of physical landscapes [1]. A landscape of dimension $n_1 \times n_2$ resulted in a graph with $n_1 \cdot n_2$ vertices and approximately $4 \cdot n_1 \cdot n_2$ edges (central vertices having four successors, "north, south, east, west"). The cost c of an edge was taken to be the difference in elevation between the destination and source vertices. The "delays" d_1 and d_2 (the second delay was used only in 3-weight graphs) were generated randomly according to a Gaussian distribution. Figures 6, 7, and 8 present the results. The notation used in these tables is explained in Figure 5. A result of the algorithm is shown in Figure 9.

From Figures 6 and 7, the following observations can be made:

- The shortest-path algorithm is two or more orders of magnitude faster than the bounded-error approximative algorithm, while at the same time producing paths which are both feasible (w.r.t. d_1) and as good as the paths produced by the bounded-error algorithm (w.r.t. c).

[1] I.e., 2-dimensional arrays, the i, j-th element giving the altitude of the longitude-latitude point corresponding to coordinates i, j.

n	Number of vertices
m	Number of edges
t	Execution time (CPU) in seconds
c	"Cost" of path
d_1	"Delay 1" of path
d_2	"Delay 2" of path
δ_ϵ	Step-error for bounded-error approximate algorithm

Figure 5: Notation for tables 6 and 7.

- The bounded-error approximative algorithm is sensitive to the step-error parameter, δ_ϵ. Reducing δ_ϵ by one or two orders of magnitude resulted in dramatic increases in running time.
- The algorithms are not very sensitive on the particular source/destination pair.

	1st source/dest. pair		
	$\delta_\epsilon = 10^{-4}$	$\delta_\epsilon = 10^{-5}$	$\delta_\epsilon = 10^{-6}$
graph 1	$t = 70$	$t = 413$	$t = 1274$
$n = 4641$	$c = 1532$		
$m \approx 4 \cdot n$	$d_1 = 2.15\%$		
graph 2	$t = 56$	$t = 1278$	$t = 16740$
$n = 36417$	$c = 4152$		
$m \approx 4 \cdot n$	$d_1 = 0.2\%$	$d_1 = 0.15\%$	
graph 3	$t = 725$	$t = 3503$	$t = 9971$
$n = 225680$	$c = 9409$		
$m \approx 4 \cdot n$	$d_1 = 0.01\%$	$d_1 = 0\%$	

Figure 6: Results of the bounded-error approximative algorithm on 2-weight graphs.

	1st source/dest. pair	2nd source/dest. pair
graph 1	$t = 0.94$	$t = 0.48$
$n = 4641$	$c = 1564$	$c = 723$
$m \approx 4 \cdot n$	$d_1 = 1.27\%$	$d_1 = 0.14\%$
graph 2	$t = 7.16$	$t = 3.48$
$n = 36417$	$c = 4220$	$c = 1184$
$m \approx 4 \cdot n$	$d_1 = 0.04\%$	$d_1 = 0\%$
graph 3	$t = 47.96$	$t = 31.56$
$n = 225680$	$c = 9411$	$c = 4487$
$m \approx 4 \cdot n$	$d_1 = 0\%$	$d_1 = 0\%$

Figure 7: Results of the shortest-path algorithm on 2-weight graphs.

From Figure 8, we see that adding one more weight/constraint to the problem dramatically increases the execution time of the bounded-error approximative algorithm, with respect to 2-weight graphs. Whereas we have been able to execute the algorithm in 2-weight graphs of size up to 225680 vertices, why we could only treat 3-weight graphs of relatively small sizes (up to 1700 vertices).

Conclusion

In this paper, we have presented several different algorithms for solving the routing problem with multiple constraints.

	$\delta_\epsilon = 10^{-4}$	$\delta_\epsilon = 10^{-5}$	$\delta_\epsilon = 10^{-6}$
graph 4	$t = 1.49$	$t = 1.80$	$t = 1.85$
$n = 777$	$c = 720$		
$m \approx 4 \cdot n$	$d_1 = 5.6\%$		
	$d_2 = 6.43\%$		
graph 5	$t = 4.25$	$t = 153.97$	$t = 6095.42$
$n = 1178$	$c = 994$		
$m \approx 4 \cdot n$	$d_1 = 1.79\%$		
	$d_2 = 5.39\%$		
graph 6	$t = 1352.54$	$t = 17631.51$	$t = 29274.12$
$n = 1722$	$c = 1024$		
$m \approx 4 \cdot n$	$d_1 = 3.68\%$		
	$d_2 = 4.66\%$		

Figure 8: Results of the bounded-error algorithm on 3-weight graphs.

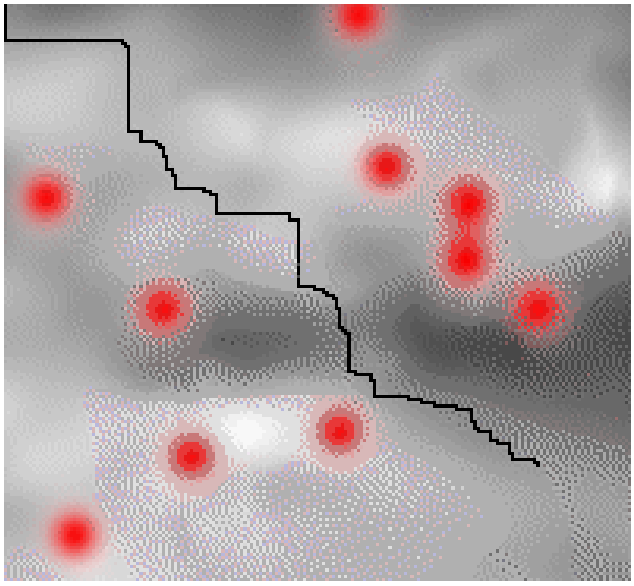


Figure 9: An output of the bounded-error approximative algorithm on a map translated into a 2-weight graph: the solid black line depicts the path; red dots are "high-delay" zones; the grey scale background represents the elevation variations of the landscape (white: high, black: low).

These algorithms vary in their complexity and the accuracy of their solutions. Our contributions mainly consist in improvements in the complexity of previously existing algorithms. We have also implemented our algorithms and examined their performance on different graphs of relatively large size.

References

S. Chen and K. Nahrstedt 1998a. On Finding Multi-Constrained Paths, *International Conference on Communications (ICC'98)*.

S. Chen and K. Nahrstedt 1998b. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions, *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*.

M.R. Garey and D.S. Johnson 1979. *Computers and Intractability — A Guide to the theory of NP-Completeness*, Freeman, San Francisco.

R. Hassin 1992. Approximation Schemes for the Restricted Shortest Path Problem, *Mathematics of Operations Research*, Vol. 17, No. 1.

J. M. Jaffe 1984. Algorithms for Finding Paths with Multiple Constraints, *Networks*, Vol. 14, pp. 95-116.

A. Orda, Routing with End-to-End QoS Guarantees in Broadband Networks 1999. *IEEE/ACM Transactions on Networking*.

H. F. Salama et. al. 1997. A Distributed Algorithm for Delay-Constrained Unicast Routing", *IEEE INFOCOM'97*, Kobe, Japan.

Z. Wang and J. Crowcroft 1996. Quality-of-Service Routing for Supporting Multimedia Applications, *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 7.

Integration of a Multicriteria Decision Model in Constraint Programming

F. Le Huédé[1,2], P. Gérard[1], M. Grabisch[1,2], C. Labreuche[1] and P. Savéant[1]
1. THALES Research and Technology FRANCE, domaine de Corbeville, 91401 Orsay cedex
2. Laboratoire d'Informatique de Paris VI, université Pierre et Marie Curie (UPMC) 8, rue du Capitaine Scott 75015 Paris
{philippe.gerard;christophe.labreuche;fabien.lehuede;pierre.saveant}@thalesgroup.com
Michel.Grabisch@lip6.fr

Abstract

In this paper we propose an integration scheme for the modeling of preferences with a fuzzy measure and the Choquet integral in Constraint Programming (CP). In order to use the Choquet integral as an objective function in CP, we define the $Choquet$ global constraint and present the principles of the algorithms that can be used to enforce arc-B-consistency on this constraint during the search. Finally we give some preliminary results of the propagation of the $Choquet$ constraint on the examination timetabling problem.

Introduction

Multicriteria Decision Making (MCDM) models subjective preferences in order to automate the determination of a preferred solution out of a set of alternatives. Constraint Programming (CP) solves combinatorial optimization problems and provides methods to implicitly explore the solutions space and determine an optimal solution with respect to an objective function. Our objective is to combine the preference modeling capacities of MCDM methods with CP search techniques. Indeed, from the multicriteria point of view, the set of alternatives to evaluate is sometimes implicitly described and its construction can be extremely complicated by the complexity of the problem or the size of the set. On the other hand, CP has been designed for the solving of large combinatorial problems. However, classical objective functions do not permit to model complex subjective preference.

In this paper, we propose the $Choquet$ constraint to integrate preference modeling with fuzzy measures and the Choquet integral in a CP solver. In order to be able to tackle large combinatorial problems, we introduce a propagation algorithm to reduce the domains of the variables involved in the constraint and thus, reduce the search space. A first evaluation of this constraint is realized on the examination timetabling problem.

Background

In this section, we introduce some basics on preference modeling with fuzzy measures and the Choquet integral as well as the main principles of Constraint Programming.

Preference modeling with fuzzy measures and the Choquet integral

During the last decades more and more interest has been given to multicriteria decision models. In this paper we focus on the fuzzy measure based model first introduced by Sugeno (Sugeno 1974). This model uses the MultiAttribute Utility Theory (MAUT) framework (Keeney & Raiffa 1976). It aims to represent interaction phenomena among criteria thanks to fuzzy measures. The overall evaluation of a solution is realized by the aggregation of the criteria and the fuzzy measure with the Choquet integral.

The MAUT framework

The MultiAttribute Utility Theory methodology is mainly concerned with the construction of additive utility functions.

Let us denote by $\mathcal{N} = \{1, \ldots, n\}$ the set of criteria. We suppose to have a set of *solutions* or *alternatives* $\mathcal{S}$ among which the decision maker must choose. Each solution is associated with a vector $x \in \Omega$ of which components $x_i \in \Omega_i, i \in \{1, \ldots, n\}$, represent the points of view to be taken into account in the decision making process. A component x_i is called *attribute* of a solution. Typically, in a multiobjective optimization context, each attribute would correspond to the value of an objective function. The modeling of the decision maker preferences $\succeq$ is realized through an overall utility function $u : \Omega \to I\!R$ such that:

$$\forall x, y \in \Omega, x \succeq y \Leftrightarrow u(x) \geq u(y).$$

Classically, this overall evaluation function is split into two parts (Keeney & Raiffa 1976):

- the *utility functions*, $u_1(x_1), \ldots, u_n(x_n)$, map each attribute to a single satisfaction scale $\mathcal{E}$ which is a real interval. This ensures *commensurateness* among criteria.
- the aggregation function aggregates the values returned by $u_1, \ldots, u_n$ and establishes the overall evaluation:

$$\forall x \in \Omega, u(x) = F(u_1(x_1), \ldots, u_n(x_n))$$

where $u_i : \Omega_i \to \mathcal{E}$ and $F : \mathcal{E}^n \to \mathcal{E}$. $u_i(x_i)$ is called *utility* or *score* of the alternative x on the criterion i. Ensuring commensurateness among criteria is an important step. However, when using compensatory aggregators (such as the weighted sum), we have to use utility values that correspond to a *scale of difference*. This means that not only

utilities must correspond to the ranking of some alternatives upon different points of view, but also the difference between two values have to make sense. This allows to treat comparable values even when the attributes of a problem are given in different units. In our study the satisfaction scale $\mathcal{E}$ is the interval $[0, 1]$. The MACBETH (Bana e Costa & Vansnick 1994) methodology is used to construct the utility functions.

The aggregation function may be of various kinds (Grabisch 1996), one of the most commonly used being the weighted sum. Nevertheless, it has to be consistent with the decision maker preferences. For instance, when using a weighted sum to model preferences, the contribution of one criteria to the overall evaluation of an alternative does not depend upon the other criteria. Actually, most of the time when an expert evaluates an alternative, he analyzes the weakest and strongest points of the alternative. The way he takes into account the satisfaction degrees depends on which criteria are well-satisfied and which are the flaws. To model this, more powerful aggregators than the weighted sum are needed.

Fuzzy measures and the Choquet integral

In order to generalize the weighted sum, (Sugeno 1974) proposes to assign weights not only to each criterion separately, but also to any coalition of criteria. These weights correspond to a set function that is called "fuzzy measure".

Definition 1 *(Fuzzy measure (Sugeno 1974))*
Let $\mathcal{P}(\mathcal{N})$ be the power set of $\mathcal{N}$. A fuzzy measure μ on $\mathcal{N}$ is a function $\mu : \mathcal{P}(\mathcal{N}) \rightarrow [0, 1]$, satisfying the following axioms.

(i) $\mu(\emptyset) = 0,\ \mu(\mathcal{N}) = 1.$

(ii) $A \subset B \subset \mathcal{N}$ *implies* $\mu(A) \leq \mu(B)$.

In this context, $\mu(A)$ represents the degree of importance of the subset of criteria $A \subset \mathcal{N}$. First, the fuzzy measure is established in order to complete the model of the decision maker preferences (Grabisch & Roubens 2000). Then, the uni-dimensional utilities $u_1, \ldots, u_n$ are aggregated with the Choquet integral to produce the overall evaluation of an alternative.

Definition 2 *(The Choquet integral (Choquet 1953))*
Let μ be a fuzzy measure on $\mathcal{N}$, and $u = (u_1, \ldots, u_n) \in [0, 1]^n$. The Choquet integral of u with respect to μ is defined by:

$$C_\mu(u_1, \ldots, u_n) = \sum_{i=1}^{n} u_{(i)}[\mu(A_{(i)}) - \mu(A_{(i+1)})], \quad (1)$$

where (i) indicate a permutation on $\mathcal{N}$ such that $u_{(1)} \leq \cdots \leq u_{(n)}$, $A_{(i)} = \{(i), \ldots, (n)\}$ and $A_{(n+1)} = \emptyset$.

The combined use of the Choquet integral and fuzzy measures allows very precise preference models to be built. In particular it allows to reach non supported solutions (i.e., Pareto optimal solutions that cannot be reached by a weighted sum, whatever the weights may be), and to model various decision making behaviors (tolerance, veto, etc.). Many usual examples of aggregation functions are particular cases of the Choquet integral, e.g., weighted sum, min, max, ordered weighted sum (Grabisch 1995).

Numerous practical applications and theoretical works (Marichal 1998) have shown that fuzzy measures combined with the Choquet integral were particularly appropriate to aggregate utilities in a multicriteria decision problem.

Basics of Constraint Programming

The *Constraint Programming (CP)* paradigm has been developed in order to model and solve combinatorial optimization problems (Van Hentenryck 1989). CP languages provide a natural framework to model a problem with *variables*, each one associated with a set of possible values called the *domain*, and *constraints*, which express relations between the variables. The solving process uses a branch and bound approach where the solutions space is reduced at each node of the search tree thanks to propagation algorithms.

The Constraint Satisfaction Problem

CP uses the *Constraint Satisfaction Problem (CSP)* (Mackworth 1977) framework to model combinatorial optimization problems.

Definition 3 *(Constraint Satisfaction Problem)*
Let $\mathcal{P} = (X, D, C)$ be a CSP. It is defined by:

- *a set $X = \{x_1, \ldots, x_m\}$ of m variables,*
- *a set $D = \{d_1, \ldots, d_m\}$ of m finite domains (usually, intervals over integers). Each variable x_i is associated with its domain d_i,*
- *a set $C = \{c_1, \ldots, c_p\}$ of p constraints between the variables.*

A constraint on a set of variables expresses which combinations of values are allowed for the variables. It can be defined either intentionally (e.g., using a predicate such as $x < y$) or extensionally by the set of tuples that satisfy the constraint. A solution to a CSP is an assignment of all variables such that all constraints are satisfied. Once an instance of the CSP has been defined, one can desire to find one solution, all solutions, or an optimal solution with respect to an objective function.

Constraint propagation

A major principle of CP is that each constraint can be actively used to deduce reductions of the search space and thus decrease the computational effort needed to solve the problem. This process is called *constraint propagation*. The CSP being $\mathcal{NP}$-complete, constraint propagation algorithms perform only partial deductions to reduce the variables domains within an acceptable computation time. Thus, constraint propagation algorithms ensure that each constraint independently of the other one is consistent with the domain of its variables. This property is called *arc-consistency*.

Definition 4 *(Arc-consistency)*
Given a constraint c over q variables $x_1, \ldots, x_q$, and a domain d_i for each variable x_i, c is said to be "arc-consistent" if and only if for any variable x_i and any value

v_i in d_i, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_q$ in $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_q$ such that $c(v_1, \ldots, v_q)$ holds.

Maintaining arc-consistency on a constraint means to remove from the domains of each variable involved in the constraint, the values for which no tuple of values can be found in the other variables domains such that the constraint is satisfied.

Global constraints (Beldiceanu & Contejean 1994; Régin 1994; Baptiste, Le Pape, & Nuijten 2001) model particular kind of constraints that are often met in combinatorial problems such as scheduling and vehicle routing problems. They implement their own propagation algorithms, exploiting the particular structure of a problem to make further deductions. In general, global constraints are based on dedicated algorithms issued from Operations Research that are particularly efficient to solve specific problems.

Consistency techniques for numeric CSPs

Numeric CSPs are special cases of the CSP where the variables are associated with discrete or continuous domains. In numeric CSPs, the domain of a variable x is represented by the interval $[\underline{x}, \overline{x}]$ where $\underline{x}$ denotes its smallest value and $\overline{x}$ denotes its largest value. A common way to propagate constraints on numeric CSPs is to maintain *arc-B-consistency* (Lhomme 1993),i.e., arc-consistency restricted to the bounds of the domains.

Definition 5 *(Arc-B-consistency)*
Given a constraint c over q variables $x_1, \ldots, x_q$, and a domain $d_i = [\underline{x_i}, \overline{x_i}]$ for each variable x_i, c is said to be "arc-B-consistent" if and only if for any variable x_i and each of the bound values $v_i = \underline{x_i}$ and $v_i = \overline{x_i}$, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_q$ in $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_q$ such that $c(v_1, \ldots, v_q)$ holds.

Integrating preference modeling in CP

This section presents the scheme that has been chosen for the integration of the multicriteria model presented in section in a CP solver.

Motivation

The main objective of the study is the integration of the fuzzy measure based multicriteria model in CP in order to perform multicriteria optimization. Beyond preference modeling and problem solving, two aspects of the techniques to be integrated have to be considered:

- A strong advantage of CP is the great flexibility it offers for modeling and solving combinatorial optimization problems. This flexibility is mainly due to the separation between problem definition and problem solving.
- Various models can result from a preference modeling process and they can be very different depending on the problem.

In the framework of our integration we expect to keep as much as possible of the flexibility of CP as well as the possibility to use various models.

The multicriteria model is composed of attributes, utility functions, the Choquet integral and its associated fuzzy measure. An attribute is a feature of a solution that is used to evaluate its quality according to some point of view (criterion). For example, in scheduling, the makespan and the max tardiness are both attributes of a schedule. A utility function is constructed for each attribute to express its degree of satisfaction and to ensure the commensurateness between the criteria. Finally a criterion is defined by an attribute and a utility function which models a preference relation. The value of a utility function for a given solution is called the utility or the score of the solution upon a criterion.

We can note that:

- Attributes in MCDM generally correspond to objective functions in CP.
- The utility functions constructed by the MACBETH methodology are often piecewise affine functions.

All this leads us to propose the following integration scheme.

Integration scheme

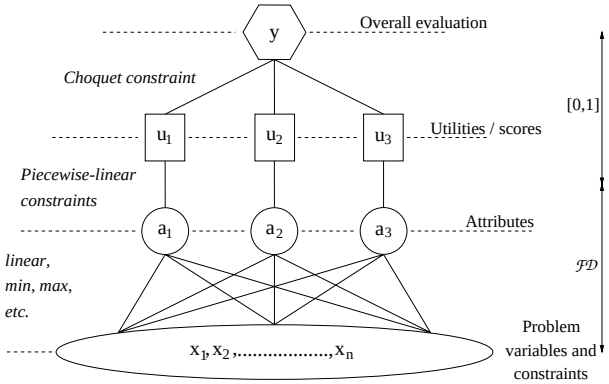


Figure 1: Integration scheme

To keep as much flexibility as possible we propose the Figure 1 example of design for the modeling of a multicriteria optimization problem. We introduce new variables:

- $y \in [0, 1]$ corresponds to the overall evaluation,
- $u_1, \ldots, u_n \in [0, 1]$ are the scores of a solution over each criterion,
- $a_1, \ldots, a_n$ are finite domain variables, they model the attributes of the problem,
- $x_1, \ldots, x_m$ are the finite domain variables of the combinatorial problem.

The combinatorial problem is modeled as usual in CP with variables and constraints. For better representation, we have extracted the attributes from the problem model. They are equal to objective functions defined on $x_1, \ldots, x_m$. On the multicriteria model side, each attribute $a_i, i \in \{1, \ldots, n\}$ is connected to a score u_i by its associated utility function. In the case of piecewise linear functions, this connection can be realized in a global constraint as described in (Milano *et al.* 2001; Refalo 1999). Finally, the variable

y to maximize must be linked with the scores $u_1, \ldots, u_n$ in order to establish the relation $y = C_\mu(u_1, \ldots, u_n)$. This relation depends on the fuzzy measure that has been calculated during the preference modeling process.

In the following, we propose the $Choquet$ global constraint to efficiently enforce this relation.

The Choquet constraint

This section presents the $Choquet$ constraint. First, we define the semantics and the signature of the constraint. Then we establish some conditions to check arc-B-consistency on the constraint. Finally, we focus on the properties of these conditions to set the basis of a propagation algorithm to maintain arc-B-consistency on the constraint during the search.

Definition

We consider the n variables $u_1, \ldots, u_n \in [0, 1]$ and $y \in [0, 1]$. We aim to establish and propagate the equality between the y variable and the Choquet integral of $u_1, \ldots, u_n$ with respect to a fuzzy measure μ. Mathematically, we want to enforce:

$$y = C_\mu(u_1, \ldots, u_n)$$

that is to say:

$$y = \sum_{i=1}^{n} u_{(i)}[\mu(A_{(i)}) - \mu(A_{(i+1)})],$$

where (i) indicate a permutation on $\{1, \ldots, n\}$ such that $u_{(1)} \leq \cdots \leq u_{(n)}$, $A_{(i)} = \{(i), \ldots, (n)\}$ and $A_{(n+1)} = \emptyset$.

To achieve this, we define the Choquet constraint:

Definition 6 *(The $Choquet$ constraint)*
Let $\mathcal{N}$ be a set of n criteria, let $\{y\} \bigcup \{u_1, \ldots, u_n\}$ be a set of variables ranging over $[0, 1]$ and let $\mathcal{M} = \{\mu(\emptyset), \mu(1), \mu(2), \ldots, \mu(1, \ldots, n)\}$ be the values of a fuzzy measure μ for each element of $\mathcal{P}(\mathcal{N})$. The $Choquet$ constraint enforces the relation $y = C_\mu(u_1, \ldots, u_n)$ and is denoted $Choquet(y, \{u_1, \ldots, u_n\}, \mathcal{M})$.

Arc-B-consistency of the Choquet constraint

Let us denote respectively $\underline{x}$ and $\overline{x}$ the minimum and maximum values of the domain of a variable x. The arc-B-consistency of the $Choquet$ constraint is given by the next proposition:

Proposition 1 *(Arc-B-consistency of the $Choquet$ constraint)*
Let $C = Choquet(y, \{u_1, \ldots, u_n\}, \mathcal{M})$ be a $Choquet$ constraint. C is Arc-B-consistent if and only if the following four conditions hold:

(1) $\underline{y} \geq C_\mu(\underline{u_1}, \ldots, \underline{u_n})$
(2) $\overline{y} \leq C_\mu(\overline{u_1}, \ldots, \overline{u_n})$
(3) $\forall k \in \{1, \ldots, n\}$:
$C_\mu(\overline{u_1}, \ldots, \overline{u_{k-1}}, \underline{u_k}, \overline{u_{k+1}}, \ldots, \overline{u_n}) \geq \underline{y}$
(4) $\forall k \in \{1, \ldots, n\}$:
$C_\mu(\underline{u_1}, \ldots, \underline{u_{k-1}}, \overline{u_k}, \underline{u_{k+1}}, \ldots, \underline{u_n}) \leq \overline{y}$

These conditions directly result from Definition 5 (arc-B-consistency).

The propagation of the $Choquet$ constraint results from the four conditions of Proposition 1. Conditions (1) and (2) allow to compute easily a new domain for y: $[max(\underline{y}, C_\mu(\underline{u})),\ min(\overline{y}, C_\mu(\overline{u}))]$. Nevertheless, deducing domain reductions for the variables $u_1, \ldots, u_n$ from conditions (3) and (4) is not straightforward.

Computing a new minimum

Here we present the calculation of the lower bound of a variable u_k, $k \in \{1, \ldots, n\}$, that can be deduced from condition (3) in Proposition 1. We denote $\check{u}_k$ this value. It is such that :

$$C_\mu(\overline{u_{-k}}, \check{u}_k) = \underline{y}$$

For sake of concision, we will use the following notations in the remaining of the paper:
$C_\mu(\underline{u}) = C_\mu(\underline{u_1}, \ldots, \underline{u_n})$
$C_\mu(\overline{u}) = C_\mu(\overline{u_1}, \ldots, \overline{u_n})$
$C_\mu(\overline{u_{-k}}, \underline{u_k}) = C_\mu(\overline{u_1}, \ldots, \overline{u_{k-1}}, \underline{u_k}, \overline{u_{k+1}}, \ldots, \overline{u_n})$
$C_\mu(\underline{u_{-k}}, \overline{u_k}) = C_\mu(\underline{u_1}, \ldots, \underline{u_{k-1}}, \overline{u_k}, \underline{u_{k+1}}, \ldots, \underline{u_n})$

Let $C_\mu^k : [0, 1] \to [0, 1]$ be the function such that $C_\mu^k(x) = C_\mu(\overline{u_{-k}}, x)$. We denote τ the permutation of $\overline{u_1}, \ldots, \overline{u_{k-1}}, \overline{u_{k+1}}, \ldots, \overline{u_n}$ such that $\overline{u_{\tau(1)}} \leq \cdots \leq \overline{u_{\tau(n-1)}}$. We will consider $u_{\tau(0)} = 0$ and $u_{\tau(n)} = 1$. In addition we define the set $A^{-k}_{\tau(i)} = \{\tau(i), \ldots, \tau(n-1)\}$.

Figure 2 gives an example of the shape of C_μ^k. It is an increasing piecewise linear function whose edges correspond to the points where x is equal to $0, \overline{u_{\tau(1)}}, \ldots, \overline{u_{\tau(n-1)}}, 1$. The first step of the calculation of $\check{u}_k$ consists in determin-

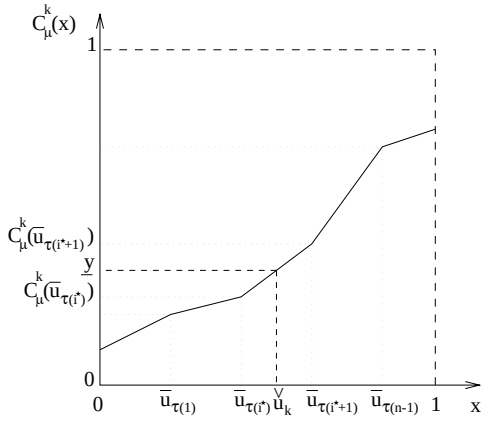


Figure 2: Shape of $C_\mu^k(x)$

ing on which piece of the curve relies the solution. This segment is defined by the integer $i^\star \in \{0, \ldots, n-1\}$, such that $C_\mu^k(\overline{u_{\tau(i^\star)}}) \leq \underline{y} < C_\mu^k(\overline{u_{\tau(i^\star+1)}})$. We are then able to deduce:

$$\check{u}_k = \overline{u_{\tau(i^\star)}} + \frac{\underline{y} - C_\mu^k(\overline{u_{\tau(i^\star)}})}{\mu(A^{-k}_{\tau(i^\star)} \cup \{k\}) - \mu(A^{-k}_{\tau(i^\star)})} \qquad (2)$$

The calculation of an upper bound $\hat{u}_k$ for the variable u_k can be derived similarly from condition (4). We are thus able to compute a new interval for any variable u_k : $[\max(\check{u}_k, \underline{u_k}), \min(\hat{u}_k, \overline{u_k})]$.

Example 1 *(Propagation of the $Choquet$ constraint)*
Let y, u_1, u_2, u_3 be four variables and let $\mathcal{M} = \{\mu_0, \mu_1, \ldots, \mu_{123}\}$ be a fuzzy measure such that:
$y \in [0.4, 1], u_1 \in [0, 0.2], u_2 \in [0, 0.8], u_3 \in [0, 0.2]$ *and*

$$\begin{array}{ccc} & \mu_0 = 0 & \\ \mu_1 = 0.1 & \mu_2 = 0.4 & \mu_3 = 0.1 \\ \mu_{12} = 0.5 & \mu_{13} = 0.2 & \mu_{23} = 0.6 \\ & \mu_{123} = 1 & \end{array}$$

If we set the constraint $Choquet(y, \{u_1, u_2, u_3\}, \mathcal{M})$, we obtain the following propagations:
Propagations on the y variable:
$C_\mu(0, 0, 0) = 0$ *and* $C_\mu(0.2, 0.8, 0.2) = 0.2 \times (1 - 0.6) + 0.2 \times (0.6 - 0.4) + 0.8 \times 0.4 = 0.44$. *Therefore from condition* (2) *of Proposition 1 we can conclude that* $y \in [0.4, 0.44]$.
Propagations on u_1:
Let $\check{u}_1$ be the lower bound of u_1 that can be deduced from Proposition 1, condition (3)*, i.e., such that* $C_\mu(\check{u}_1, 0.8, 0.2) = 0.4$. *We have already calculated* $C_\mu(0.2, 0.8, 0.2) = 0.44$. *Therefore we can conclude that* $\check{u}_1 \in [0, 0.2)$ *and from Equation (2):*

$$\check{u}_1 = \frac{0.4 - C_\mu(0, 0.8, 0.2)}{\mu_{123} - \mu_{23}} = \frac{0.4 - 0.36}{0.4} = 0.1$$

Considering the calculation of an upper bound for u_1, we can notice that we have reduced $\overline{y}$ such that $\overline{y} = C_\mu(\overline{u_1}, \overline{u_2}, \overline{u_3})$. It follows that $\overline{y} \geq C_\mu(\overline{u_1}, \underline{u_2}, \underline{u_3})$. Therefore condition (4) *of Proposition 1 is verified and we can conclude that $\overline{u_1}$ will not be reduced.*
If we follow the same reasoning for u_2 and u_3, we finally obtain: $u_1 \in [0.1, 0.2]$, $u_2 \in [0.7, 0.8]$ and $u_3 \in [0.12, 0.2]$.

Maintaining arc-B-consistency

Domain reduction calculations for the variables $u_1, \ldots, u_n$ have a relatively high computation cost with respect to the frequency of their invocation. For the calculation of new lower bounds, the variables $\overline{u_1}, \ldots, \overline{u_n}$ have to be sorted and the Choquet integral may have to be computed several times. However, we have seen in Example (1) that it was possible to save some calculus, either by re-using a value of the Choquet integral previously calculated or by identifying some relations between the conditions of Proposition 1.

In this section, we present some useful properties in order to design algorithms for the propagation of the $Choquet$ constraint and to reduce useless calculus.

Again, the following properties are deduced from the increasingness of C_μ. The objective is to show that some conditions of Proposition 1 cannot be violated simultaneously and that in consequence some bound calculations can be avoided.

The following properties show some incompatibilities on the simultaneous violation of conditions on y and conditions on $u_1, \ldots, u_n$:

(i) Condition (1) and (3) cannot be violated simultaneously. Indeed:
$\underline{y} < C_\mu(\underline{u}) \Rightarrow \forall k \in \{1, \ldots, n\}, C_\mu(\overline{u_{-k}}, \underline{u_k}) \geq \underline{y}$
and
$\exists k \in \{1, \ldots, n\}, C_\mu(\overline{u_{-k}}, \underline{u_k}) < \underline{y} \Rightarrow \underline{y} \geq C_\mu(\underline{u})$.

(ii) Similarly, condition (2) and (4) cannot be violated simultaneously:
$\overline{y} > C_\mu(\overline{u}) \Rightarrow \forall k \in \{1, \ldots, n\}, C_\mu(\underline{u_{-k}}, \overline{u_k}) \leq \overline{y}$
and
$\exists k \in \{1, \ldots, n\}, C_\mu(\underline{u_{-k}}, \overline{u_k}) > \overline{y} \Rightarrow \overline{y} \leq C_\mu(\overline{u})$.

A less trivial property shows the same kind of incompatibilities on conditions on different score variables:

(iii) For all pair of distinct variables u_i, u_j, condition (3) on u_i and condition (4) on u_j cannot be violated simultaneously. That is to say, $\forall i, j \in \{1, \ldots, n\}, i \neq j$, we have necessarily:
$C_\mu(\overline{u_{-i}}, \underline{u_i}) < \underline{y} \Rightarrow C_\mu(\underline{u_{-j}}, \overline{u_j}) \leq \overline{y}$
and
$C_\mu(\underline{u_{-j}}, \overline{u_j}) > \overline{y} \Rightarrow C_\mu(\overline{u_{-i}}, \underline{u_i}) \geq \underline{y}$.

In CP, propagation algorithms are said to be "event driven": the constraint is awaken by a modification on the domain of one of its variable and triggers an appropriate algorithm, depending on the kind of modification (increasing of a lower bound, decreasing of an upper bound, removing of a value from a domain, assignment of a variable to a value). Properties (i), (ii) and (iii) allow to trigger only appropriate calculus for the propagation of the Choquet constraint and establish necessary and sufficient conditions to ensure that arc-B-consistency is maintained although not all conditions of Proposition 1 have been verified.

Experimentations

The $Choquet$ constraint has been implemented in the Eclair solver (Laburthe *et al.* 1998) and has been evaluated on the examination timetabling problem.

The examination timetabling problem

Given a set of examinations, a set of students each enrolled to a given list of examinations, a set of rooms of fixed capacities, and a set of periods, the examination timetabling problem consists in assigning a period and a room to each examination such that (i) two examinations that are given to a same student cannot be planned on the same period and (ii) the capacity of a room cannot be exceeded. We assume that as long as constraints (i) and (ii) hold, several examinations can occur in the same room at the same time but that the number of students attending an examination cannot be distributed over several rooms.

A simple multicriteria model has been constructed based on three attributes: the total duration of the examinations, the number of rooms used and the number of students that have two consecutive examinations.

Data sets

Small scenarios have been constructed in order to evaluate the pruning realized by the constraint. These scenarios are briefly described in the following table:

	Number of periods	Number of exams	Number of rooms	Number of students
Sc. 1	6	7	2	5
Sc. 2	6	7	2	29
Sc. 3	9	10	3	38

Results

The results presented here compare the solving of the scenarios using a basic propagation from the scores u_1, u_2, u_3 to the overall evaluation y, to a solving where arc-B-consistency is enforced at each node of the search tree. To have a good vision of the impact of the propagation on the performance of the algorithm, the search tree is constructed using a static chronological labeling strategy. The variables that model the date of the exams are ordered following the number of disjunctions between exams and the number of enrollments to each exam. The performance of the algorithm is given by the number of backtrackings realized during the search and by the completion time.

	Basic propagation		Interval consistency	
	Number of backtracks	Resolution time (ms)	Number of backtracks	Resolution time (ms)
Sc. 1	68	30	63	20
Sc. 2	287	130	145	120
Sc. 3	41594	17300	15837	8600

As can be seen for the first and second sets of data, maintaining arc-B-consistency may be costly on small problems although the constructed search tree is smaller. Nevertheless, the propagation seems to be advantageous when considering larger problems. Note that in both cases, the solving seems to be long with respect to the size of the problems. However, a quite naive approach has been used to model room capacities. This could be greatly improved using the Cumulative global constraint as described in (Boizumault *et al.* 1995), but this constraint is not yet available in Eclair. Furthermore, when handling multiple contradictory criteria, it is highly probable that with a standard labeling strategy, good solutions are spread over the search tree. Our future work will focus on the development of specific labeling strategies that could perform well in a multicriteria optimization context.

Conclusion

In this paper we proposed an integration scheme for the modeling of preferences with a fuzzy measure and the Choquet integral in CP. In order to use the Choquet integral as an objective function in CP, we defined the $Choquet$ global constraint and presented the principles of the algorithms used to enforce arc-B-consistency on this constraint during the search. Finally some preliminary results of the propagation of the $Choquet$ constraint were given to solve the examination timetabling problem. We can observe that although the constraint propagation performs a considerable pruning, still a wide area of the search tree is explored. This is usually tackled in constraint programming by defining efficient labeling strategies in order to quickly guide the search toward good solutions. The next step of our study will consist in determining how the preference model can help to the construction of such strategies.

References

Bana e Costa, C. A., and Vansnick, J. C. 1994. A theoretical framework for Measuring Attractiveness by a Categorical Based Evaluation TecHnique (MACBETH). In *Proc. XIth Int. Conf. on MultiCriteria Decision Making*, 15–24.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-Based Scheduling - Applying Constraint Programming to Scheduling Problems*. Kluwer.

Beldiceanu, N., and Contejean, E. 1994. Introducing global constraints in CHIP. *Mathematical and Computer Modelling* 12:97–123.

Boizumault, P.; Delon, Y.; Guéret, C.; and Péridy, L. 1995. Résolution de problèmes en programmation logique avec contraintes. *Revue d'intelligence artificielle* 9(3):383–406.

Choquet, G. 1953. Theory of capacities. *Annales de l'Institut Fourier* 5:131–295.

Grabisch, M., and Roubens, M. 2000. Application of the choquet integral in multicriteria decision making. In Grabisch, M.; Murofushi, T.; and Sugeno, M., eds., *Fuzzy Measures and Integrals — Theory and Applications*. Physica Verlag. 348–374.

Grabisch, M. 1995. Fuzzy integral in multicriteria decision making. *Fuzzy Sets & Systems* 69:279–298.

Grabisch, M. 1996. The application of fuzzy integrals in multicriteria decision making. *European J. of Operational Research* 89:445–456.

Keeney, R. L., and Raiffa, H. 1976. *Decision with Multiple Objectives*. New York: Wiley.

Laburthe, F.; Savéant, P.; de Givry, S.; and Jourdan, J. 1998. ECLAIR: A library of constraints over finite domains. Technical Report ATS 98-2, Thomson-CSF LCR, Orsay, France.

Lhomme, O. 1993. Consistency techniques for numerical CSPs. In *Proceedings of IJCAI 1993*, 232–238.

Mackworth, A. K. 1977. Consistency in network of relations. *Artificial Intelligence* 8:99–118.

Marichal, J. L. 1998. *Aggregation operators for multicriteria decision aid*. Ph.D. Dissertation, University of Liège.

Milano, M.; Ottosson, G.; Refalo, P.; and Thorsteinsson, E. 2001. Global constraints: When constraint programming meets operation research. *INFORMS Journal on Computing, Special Issue on the Merging of Mathematical Programming and Constraint Programming*. Submitted.

Refalo, P. 1999. Tight cooperation and its application in piecewise linear optimisation. In *Proceedings of Principles and Practice of Constraint Programming*, 375–389. Springer.

Régin, J. C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI94*, 362–367.

Sugeno, M. 1974. *Theory of fuzzy integrals and its applications*. Ph.D. Dissertation, Tokyo Institute of Technology.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. The MIT Press.

An Optimization Framework for Interdependent Planning Goals

Tara A. Estlin, Daniel M. Gaines

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109-8099
{firstname.lastname}@jpl.nasa.gov

Abstract

This paper describes an approach for optimizing over inter-dependent planning goals. Most planning systems allow only simple, static dependencies to be defined among goals where these dependencies remain constant between different problems. However, in many domains, goals are related through detailed utility models that may significantly change from problem to problem. For instance in one problem, a particular goal's utility may increase if other related goals can be achieved. In another problem, this utility increase may differ or actually decrease if the same combination of goals is achieved. To address these types of problem situations, we have implemented a methodology for representing and utilizing information about interdependent goals and their related utilities using the ASPEN planning and scheduling system. We show through experimental results that this approach significantly increases overall plan quality versus a standard approach that treats goal utilities independently.

Introduction

As the sophistication of planning techniques grows, these systems are being applied to an increasing number of real-world problems. Planning and scheduling techniques are currently being applied with great success to handle problems in manufacturing, logistics, and space exploration. In a typical application, a planner is given a set of goals, and it then constructs a detailed plan to achieve the goals where the plan must respect a specific set of domain rules and constraints. A limitation of most planning systems, however, is that they define relationships between input goals in a simple, static manner, which cannot be easily adjusted for different problem situations. In many domains, goals can be related in complex and varying ways that are best represented through utility metrics. These metrics are hard to include as part of a standard domain definition, since they are often dependent on current data and can vary widely from problem to problem.

When planning for NASA spacecraft or rover missions, planning goals are often dictated by science data that has just been collected. Goal utilities and dependencies for new science measurements are often dependent on a current data model and on what new science opportunities are available. Goal interdependencies can be seen in other domains as well. For instance, consider a travel-planning domain where we are planning a business trip for several people to the same location. Thus, all travelers need to arrive at the same destination and in the same general timeframe. In most cases, they would all prefer to arrive on the same day and time, however, plans that have some travelers arriving one day earlier are still valid and would still be considered. Furthermore, preferences for when people arrive could change from trip to trip. On one trip it may be important that a certain set of people arrive on the same day to attend a particular meeting. On other trips this criteria may be less important or apply to a different set of people. Representing such information in current planning systems would be difficult since most goal dependencies cannot easily change between problem instances based on new preference information.

Approaches to goal handling and representation vary widely among planning and scheduling systems. In some approaches, all goals must be achieved for the planner to even reach a solution. In other approaches, goals can be given different priorities or utilities, and the planner will try to create a plan that achieves the highest utility score where some goals may not be added to the plan. Other approaches enable a planner to accept both goals and other quality objectives, such as minimizing makespan, avoiding missed deadline costs, or minimizing the usage of a particular resource (Williamson and Hanks, 1994; Joslin and Clements, 1999; Rabideau, et al, 2000). However, even in approaches that allow the usage of more flexible optimization metrics, goal relationships are pre-defined in a domain model and typically remain relatively constant between problem instances. Furthermore, it is difficult to define utility metrics that involve specific goal instances as opposed to a general quality concept that applies to a certain class of goals (e.g., increasing the number of orders filled).

Most planning systems do allow you to define some types of *static dependencies* between goals. For instance, two goal or action types could be defined as related in a domain model, perhaps through a decomposition of a

Goal Num	Target Description	Location (x,y,z)	Reward
1	Spectrometer read for rock type x	(3.4, -34.6, 2.0)	10
2	Spectrometer read for rock type x	(162.3, 43.9, 1.1)	10
3	Spectrometer read for rock type x	(-4.1, 145.8, 0.4)	10
4	Spectrometer read for rock type y in area A	(104.3, -12.1, 1.5)	12
5	Soil sample from area A	(103.5, -13.4, 0.2)	15
6	Rock image for rock type y in area A	(104.3, -12.1, 1.5)	10
7	Dust collection experiment from area A	(105.1, -13.7, 1.5)	12

Table 1: Example sets of science goals given to planning system

parent activity. In a travel domain, you might want to tie a "board-plane" action with a "deboard-plane" action, since both will commonly occur in the same plan. Some static dependencies may also be defined automatically through other parts of the model definition. For instance, pre- and post-conditions links can relate certain goals. A domain model does typically allow goals to be linked in optional ways (e.g., a goal that could decompose to several different sets of actions or goals), however, these options are usually limited to several commonly-seen combinations. Encoding a large number of dependency options in a domain model would be intractable both for modeling ease and search complexity. No current planning systems enable *dynamic dependencies* among goals, i.e. dependencies that significantly vary from problem to problem, that can be easily utilized and defined as part of the problem specification instead of the domain model.

This paper presents a method for handling interdependent planning goals while performing plan construction and optimization. In this approach, interdependencies between goals can be formulated dynamically and provided to the planning system as part of the goal input. The planning system can then reason about these dependencies and incorporate them into the overall objective function it uses to rate plan quality and direct its search process.

This is particularly important when attempting to optimize a plan relative to multiple criteria. One approach to planning with multiple criteria is to combine the different objective functions into a single metric representing overall plan quality. However, for many domains, these objectives will interact in complex (e.g. nonlinear) ways making it difficult to improve plan quality. Our approach represents a step toward addressing this problem by providing the planner with an explicit representation of the interdependent relationships among the individual criteria that contribute to overall plan quality. Our planner uses this information to guide its search toward higher quality plans.

This optimization approach has been implemented on top of the Automated Scheduling and Planning Environment (ASPEN) (Chien, et al., 2000). ASPEN already has a base optimization framework that we have extended to handle this class of problems (Rabideau, et al., 2000). This new approach has been tested on a series of problems based on a team of rovers performing geological experiments in a new terrain. Even with our current implementation's relatively simple objective function and search technique, experimental results show that by using information about related goals, our approach is able to significantly improve plan quality.

Planning for a Multi-Rover Domain

In recent years, NASA has begun to focus on missions that utilize rovers to perform exploration and understanding of planetary terrains. Future missions will likely send teams of rovers to autonomously explore planetary surfaces.

To produce plans for a team of rovers, we have adapted a version of the ASPEN planning system (Estlin, et al., 1999). ASPEN automatically generates the necessary activity sequence to achieve a set of input goals. One of the main algorithms used to produce this sequence is a local, early-commitment version of iterative repair (Minton and Johnston, 1988; Zweben et al., 1994), which classifies plan conflicts and attacks them individually. For the experiments presented in the paper, planning is performed in a centralized fashion, where one planner controls multiple rovers. In future work, these techniques will be migrated to operate in a distributed planning system, where each rover has a separate onboard planner controlling its operations (Estlin, et al., 2000).

Plan Optimization

ASPEN provides an optimization framework that allows the representation of continuous *soft constraints* (i.e., preferences) (Rabideau, et al., 2000). In contrast to traditional *hard constraints*, soft constraints do not have to be satisfied for the plan to be valid. However, satisfying them will improve the quality score for the plan.

In ASPEN, a preference is defined as a mapping from a plan variable (e.g. resource level, goal count, etc.) to a quality metric. Specifically, a preference indicates whether the score is monotonically increasing or decreasing with respect to the plan variable. The overall plan score is the weighted sum of individual preference scores.

An iterative optimization algorithm, similar to iterative repair, is used to improve plan quality. For each defined preference, an improvement expert automatically generates modifications that could potentially improve the preference score. In the following sections we illustrate how we extended ASPEN's optimization framework to deal with interdependent goal combinations.

Interdependent Goals and Utilities

Historically in planning and scheduling systems, goal selection has been a linear process in which goals are independently selected and prioritized based on their expected reward. However, in some applications, this model is insufficient to correctly characterize the utility of a plan. For instance, in the case of performing science experiments in a new planetary terrain, goal priorities should be determined by the expected scientific gain, which is dependent on data already collected and available science targets. There are many situations in this type of domain where the value of a science goal will be increased if other related science goals can also be achieved. For instance, collecting images of a particular rock from different angles and distances often increases the value of all images taken of that rock, since a better overall analysis of the rock can be done. Conversely, there are situations in which it is very important to achieve one of a set of goals, but having accomplished one in the set, the others become less important. For instance, we may want a rover to collect one or two more samples of a particular rock type but there are a large number of possible targets from where to collect such a sample. In this situation, we would like to direct the planner to collect a couple samples and then move on to other science experiments. If samples were collected at all target sites, this data would be overly redundant and somewhat lower the utility of the overall set since time had been wasted collecting unneeded data.

To represent a goal's value, we have extended a typical goal-utility representation (where goals can have individual rewards representing their importance) so that complex interdependencies and their relevant utilities can be represented and utilized by a planning system. Furthermore these interdependencies and utilities can change between problem specifications without requiring any changes to the planner domain model. In our representation, a list of goals and goal combinations are provided to the planner. A utility value is also assigned to each goal and to each specified goal combination. As an example, consider the spectral measurement and image goals shown in Table 1, which are from the previously introduced rover domain. Let's assume these goals are interdependent in several ways. First, Goals 1-3 are for spectrometer readings for the same type of rock and it has been deemed necessary to obtain only one such reading and any more would add little value to the current set of collected data. Second, Goals 4-7 are for the same rock or rock area and it has been determined desirable to obtain all of those observations. However, if only a few can be obtained that data would still be beneficial but not provide as much scientific value as the entire set.

These types of goal combinations are difficult to represent in standard planning-optimization approaches. As mentioned previously, a number of systems represent goal rewards in the form of utility functions or preferences, however, these approaches typically try to maximize a certain goal type or minimize usage of a certain resource. For instance, a utility function may try to minimize the amount of fuel used in transporting objects, or may try to maximize the number of factory orders that can be filled. This type of representation is limited in that it prefers to decrease or increase the number of goals or activities of a general type, where each goal or activity is viewed as relatively equal (or interchangeable). The goal inter-dependencies required for deducing many scientific hypotheses are often much more complex since each individual goal may play a different role in the overall success of an experiment.

We can visually represent goal inter-dependencies between a set of two goals by using a graph structure where vertices represent individual goal rewards and edges represent interdependent goal rewards. For example, Figure 3 shows two goals that have individual rewards (represented by G_1 and G_2) and a combined reward (represented by R_{12}). There may also be dependencies between larger sets of goals, and thus the graph may contain hyperedges linking several goals to their combined value. Table 2 shows interdependent goal rewards for the goals introduced in Table 1. Goal combinations for goals 1-3 are given slight negative rewards to show that achieving more than one goal in this set actually has less value than just achieving one. The goal combination for goals 4-7 shows that achieving all of the goals in that set has a large bonus reward.

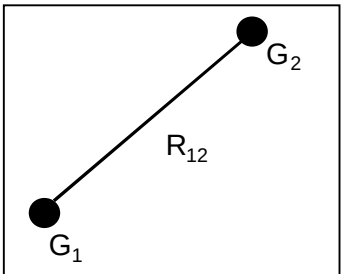


Figure 3: Two related goals

Plan Optimization for Interdependent Goals

We extended the ASPEN optimization system to support the inclusion of goal interdependencies with a planning problem description. The extension consists of two main components: an objective function to compute the value of the plan with respect to the goal interdependencies and an optimization framework for selecting goals to achieve and coordinating optimization with plan repair.

Objective Function

As is the case with most planners, the ASPEN problem specification includes a description of the goals that must be achieved to accomplish a particular problem. In

Goal Combination	Reward
<Goal 1, Goal 2>	-5
<Goal 1, Goal 3>	-5
<Goal 2, Goal 3>	-5
<Goal 4, Goal 5, Goal 6, Goal 7>	60

Table 2: Goal interdependencies and corresponding rewards

addition, ASPEN can accept a set of optional goals that, while not required, will increase the quality of the plan as more of these goals are accomplished. This is useful when the planner is given more goals than are feasible to achieve given its resource constraints. In this case, ASPEN will use an objective function to try to find a subset of goals that result in a valid, high quality plan.

Our extended version of ASPEN also takes as input a set of goal interdependencies specified as a graph of goal nodes as described in the previous section. The graph consists of a set of vertices V where each vertex corresponds to a goal that can be added to the plan, including both mandatory and optional goals, and a set of edges E. Each edge consists of a tuple of vertices: $<v_1, v_2, \ldots v_n>$. For each vertex and each edge, there is an associated weight $w_{<v1, v2, \ldots vn>}$ indicating the value that will be added to the plan if the plan includes these goals. This representation allows us to express singleton goal values, that is a goal whose contribution to the plan does not change as other goals are added, and any n-ary goal relationship to indicate the value that combination of goals add to the plan.

We use a simple objective function to calculate the plan quality with respect to these optional goals. Let G be the set of goals that occur in the plan. The value of plan P is then given by Equation 1. This function sums up the values of all goals that occur in the plan along with the weight for each edge for which all of the edge's vertices occur in the plan.

$$O(P) = \sum_{v \in V} o(\prec v \succ) + \sum_{\prec v_1, v_2, \ldots, v_n \succ \in E} o(\prec v_1, v_2, \ldots, v_n \succ)$$

$$o(\prec v_1, v_2, \ldots, v_n \succ) = \begin{cases} w_{\prec v_1, v_2, \ldots, v_n \succ} \text{ if } \{v_1, v_2, \ldots, v_n\} \subseteq P \\ 0 \text{ otherwise} \end{cases}$$

Equation 2: Objective function for calculating plan utility when using interdependent goals

Optimization Framework

The next step is to provide an improvement expert that can suggest what changes ASPEN should make to the plan to increase this score. Clearly, the improvement expert for interdependent goals should suggest adding more optional goals to the plan. However, adding a goal will likely result in conflicts in the plan. Therefore it is also necessary to coordinate the process of improving the plan score with ASPEN's repair process to fix conflicts in plans.

Our current approach to performing optimization for interdependent goals is randomized hill-climbing with restart. We begin by first creating a plan that achieves all of the mandatory goals. We then perform a series of optimization steps where each step consists of *i* iterations. At each iteration, if there are no conflicts in the plan, we use the improvement expert to suggest the next optional goal to add. If there are conflicts, we perform an iteration of repair. Whenever we have a conflict free plan, if its score is the best we have seen, we record its point in the search space. At the end of the *i*th iteration, we return to the highest-valued point in the search space and begin the next optimization step. This approach protects against the possibility of adding a goal to the plan that cannot be solved.

We use a simple, greedy improvement expert to select the next goal to add. It considers all goals and picks the one that would lead to the highest score if it were added to the plan. We include an element of randomness to avoid repeatedly adding an unachievable goal. With probability 1 - ε we add the highest scoring goal, otherwise a goal is picked at random.

Evaluating ASPEN's Performance with Interdependent Goals

Our main concern in evaluating our system was to see whether or not explicitly taking into account goal interdependencies during optimization would significantly improve the quality of the plan. We expected to see some improvement over a system that did not use goal interdependencies, but were not sure if the improvement in quality would be worth a potential increase in time to produce the plans. We were also curious to see how much of an improvement would be provided by our relatively simple objective function.

Methodology

We compared our extended version of ASPEN, which we will refer to as ASPEN+IDGS (for ASPEN with InterDependent Goal Support) to two other versions of ASPEN: ASPEN+Random and ASPEN+SimpleReward. All three versions used the randomized hill-climbing algorithm described in the previous section. The only difference is in how each of the three selects the next optional goal to add to the plan. ASPEN+IDGS uses the objective function from Equation 1 to pick the next goal. ASPEN+Random simply selects a goal at random without considering rewards. Finally, ASPEN+SimpleReward uses an objective function that looks at rewards for individual goals without considering goal interdependencies.

We ran each system on a set of randomly generated problems from a Mars exploration domain. In this domain, a team of three rovers must collect different types of science data at various locations on the planet's surface.

The planner must decide which goals to assign to each rover, determine a sequence for each rover to use in visiting the different locations, and plan for activities such as manipulating the rover masts and communicating with earth. Generated plans must also respect resource and temporal constraints, such as not exceeding onboard memory limitations when collecting data.

The randomly generated problems varied in the number and location of the science goals. Table 3 shows the types of goals that are given to the planner along with the possible rewards for each individual goal. Note that some goals have a range of rewards in which case a specific reward is drawn randomly from this range. Each problem specification contains several mandatory panoramic images (goal type A) of different terrain areas, which always provide a base set of data on each area, and then a set of optional goals to take additional images and spectrometer measurements (goal types B, C, and D) of particular rocks in those areas. Problems could range in size from 6 to 78 different goals to examine 0 to 24 rocks in the surrounding terrain.

Goal	Reward
A: Panoramic Image of an Area (Mandatory)	20
B: Long-Range Image of a Rock	12-25
C: Close-Up Image of a Rock	7-20
D: Close-Up Spectrometer Read of a Rock	2-15

Table 3: Individual goals and rewards

The rovers are given 1 Martian day to complete these goals. Depending on the relative locations of the targets, each rover can typically handle about 10 goals in this time. With three rovers this means that most of the problems will be too large to complete and the planner will have to take into account the different goal values to determine which goals should be achieved.

Each problem description also included a randomly generated set of goal interdependencies. Although the interdependencies were randomly generated, they were based on preferences derived from our conversations with planetary geologists and represent the type of utility values considered by human experts. Table 4 shows the goal combinations used for the experiment and the associated rewards. To increase the variance among goal combinations, we used two different factors for computing the value for one of the goal pairs (pair B and D). A certain percentage of the time the reward for this pair was significantly increased. Finally, for a given rock, each of the three goal combinations is removed with probability 0.5.

In selecting parameters for the randomized hill-climbing algorithm used in each planner, we decided to use 50 iterations per optimization step as it seemed to provide the best balance between allowing the planner enough time to repair goals but not so long that it would waste a lot of time if it got stuck and needed to back up to a previous plan. For ε, we selected a small value of 0.02.

Goal Combination	Reward
<Goal B, Goal C>	(Reward(B) + Reward(C)) * 1.75
<Goal B, Goal D>	(Reward(B) + Reward(C)) * 2.25, 90% (Reward(B) + Reward(C)) * 10.0, 10%
<Goal C, Goal D>	(Reward(C) + Reward(D)) * 1.25

Table 4: Goal interdependencies and rewards

Results

We generated a set of 30 problems and because there is an element of randomness both to the ASPEN iterative repair algorithm and to our optimization approach, we ran the three versions of ASPEN on each problem 5 times. The systems were run on a Sun Blade 1000 with 1 Gigabyte of RAM.

At the end of each optimization step we recorded the current plan score based on the objective function from Equation 1, the current number of goals in the plan, and the number of seconds spent during that step. Note that even though the ASPEN+Random and ASPEN+SimpleReward versions of the planner did not make use of the objective function to select goals to add, we still used that objective function to score their plans for the purpose of the experiment.

Figures 4-6 present the results from these runs. Objective function scores are compared in Figure 4, while Figures 5 and 6 compare the total number of goals achieved and the planning time used by each method. The data points in each graph are averaged over the 150 runs from each system. In each graph, the data point at optimization step 0 represents the planner performing repair on a plan containing all mandatory goals. We performed two-tailed t tests between each pair of the three systems with a Bonferroni correction. The only graph that showed significant differences among the systems was the graph of plan scores in Figure 4. ASPEN+IDGS was found to be significantly better than both ASPEN+Random and ASPEN+SimpleReward at the 0.01 confidence level. ASPEN+Random outperformed ASPEN+SimpleReward but only the data points between optimization steps 6 and 14 showed significant difference at confidence level 0.01.

Discussion

Figure 4 shows that ASPEN+IDGS outscores both ASPEN+Random and ASPEN+SimpleReward. In fact, ASPEN+IDGS showed a significant improvement over both versions at each data point. The plot of the number of goals included in each plan (Figure 5) shows that all three systems were achieving about the same number of goals. This means that ASPEN+IDGS was selecting higher quality goals. This factor is particularly important because none of the planners were able to achieve all of the goals thus it is better to achieve the higher quality subset.

It is also important to note that ASPEN+IDGS's biggest improvements in performance occur in the early

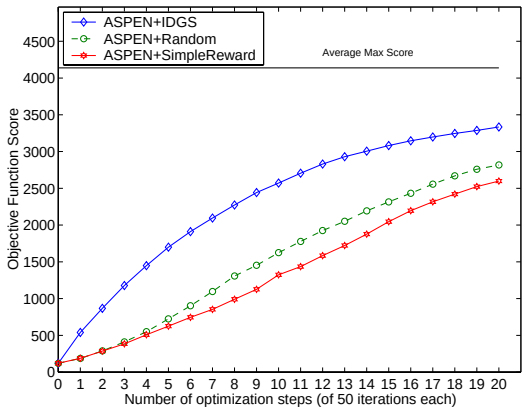


Figure 4: Objective function score

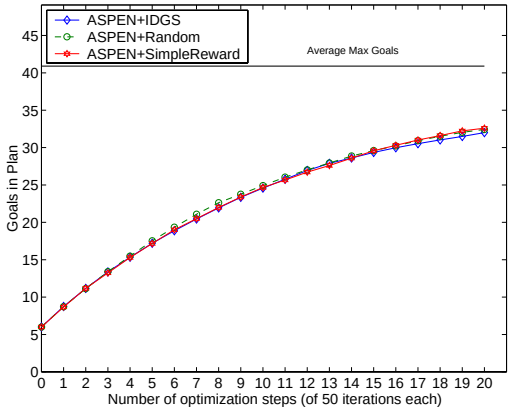


Figure 5: Number of goals achieved

optimization steps. Thus, even if the planner is capable of solving all the goals it is given but it is under tight time constraints, then using ASPEN+IDGS will allow the planner to find a much higher quality set of goals. This feature is especially important in real-world problems where planning time can be tightly bound.

The shapes of the curves reveal some interesting characteristics about each algorithm. The curve for ASPEN+IDGS rises sharply in the early optimization steps and then tapers off, while ASPEN+Random starts rising more slowly, increases in its rate of growth, and then begins to taper off at the end. Given that both planners were adding about the same number of goals to the plan at each time step, the differences in the curve shapes is a result of the way each algorithm selected goals. The sharp rise in the ASPEN+IDGS curve can be explained by the fact that ASPEN+IDGS is explicitly looking to add goals that will improve the objective function. However, as more goals are added to the plan, and therefore the rovers' resources are beginning to be stretched to their limit, making repairs to the plan becomes more difficult and the planner spends more iterations fixing problems with the plan and fewer iterations adding goals. As a result, the curve begins to level off. As can be seen in Figure 5, the number of goals added to the plan at each optimization step begins to decrease at about the same time that ASPEN+IDGS's score begins to taper off in Figure 4.

In contrast, the ASPEN+Random curve in Figure 4 begins slowly because it is randomly adding goals to the plan and, early on, it is unlikely that the interdependent goal combinations will be satisfied in the plan. However, as more goals are added, the probability of satisfying goal combinations when a new goal is added increases, and the score begins to rise more rapidly. But, just like ASPEN+IDGS, the planner begins to spend more time performing repairs and fewer goals are added to the plan causing the curve to taper off.

The fact that ASPEN+SimpleReward was the worst performer is particularly interesting. Recall that this version of the system is selecting new goals based on the each goals individual contribution to the plan. In other words, it is using the rewards from Table 3. Therefore, the planner will favor the addition of long-range images and avoid adding close-up spectrometer reads. The problem with this approach is that the goal interdependencies do not necessarily preserve the relative reward values of the individual goals. For example, although the close-up spectrometer read is the lowest rank score individually, when it is combined with a long-range image, it becomes much more valuable. However, since ASPEN+Simple-Reward typically avoids adding this goal to the plan, it does not satisfy these high-quality goal combinations. As a result, its score grows slowly and, like the other curve, tapers off in later optimization steps.

Figures 4 and 5 show that ASPEN+IDGS provides considerable benefit when the planner cannot achieve all the goals in a plan. In this case, ASPEN+IDGS selects a higher quality subset of goals than either of the two competing systems in this study. This is already advantageous, but we were also interested in whether or not ASPEN+IDGS could increase plan quality without a significant increase in planning time. The plot of each system's processing time per optimization step in Figure 6 shows ASPEN+IDGS did not significantly increase planning time.

These results show that ASPEN+IDGS provides a significant improvement in plan score over versions of the planner that do not consider goal interdependencies without a significant increase in planning time. This benefit is most important when a planner is given more goals than it can achieve as well as when the planner is under time constraints and may not have enough time to plan for all of its goals.

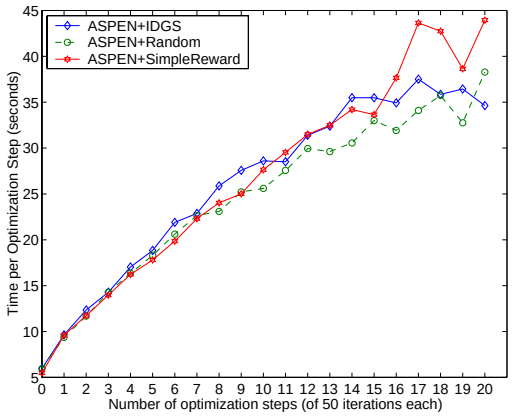


Figure 6: Plan generation time

Related Work

Other work in planning optimization has used utility models to improve on particular types of quality measures. PYRRHUS (Williamson and Hanks, 1994) extends the UCPOP partial-order planner to handle metric time, resources, and a utility model. In contrast to PYRRHUS, our approach allows for the representation of utility for specific goal combinations that can change from problem to problem.

Markov Decision Processes (MDPs) (Boutilier, et al., 1999) represent another approach to dealing with plan quality. The goal combinations used in this paper could be encoded into an MDP. However, MDPs have yet to be demonstrated on real problems of significant size in domains with time and resource constraints and it is likely that the large computational cost would be prohibitive.

Work in mixed-initiative planning allows a planner to be biased toward solutions with certain characteristics (Myers and Lee, 1999). While our work has focused on automated planning, a user could specify utility preferences to encourage certain goal combinations.

Previous work in decision analysis has looked at decision making with multiple objectives (Keeney and Raiffa, 1993) enabling one to develop preferential structures over decision outcomes. Our representation of goal interdependencies is a simple type of preference structure which allows the planner to select among alternate actions. In the future we plan to incorporate more results from decision analysis to support more complex goal relations and uncertainty about goal pay-off.

Conclusions

In this paper we have presented a method for utilizing interdependent goal utilities, where goal relations can be dictated by current information and can vary from problem to problem. In typical planning systems, only simple, static goal relations can be defined that remain relatively constant between problem instances. However, in many application areas, goal dependencies and their related utility metrics can dramatically change based on current information or even user preferences. To address this problem, we have implemented a new method for representing and reasoning about interdependent goals. We have also presented experimental results that show how this approach can significantly improve overall plan quality in a multi-rover application.

In future work we will consider more complex goal interdependencies including relations among more than two goals, relations in which only so many of a certain set of goals should be achieved, and situations in which adding certain combinations of goals can decrease plan quality. We also plan to enhance our current optimization algorithm to better recognize potential high-utility goal combinations. Finally, though currently this system is operated only in simulation, we intend to ultimately test its capabilities using real rovers examining actual terrain features.

Acknowledgments

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

Boutilier, C., Dean, T. and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1-94.

Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., and Tran, D. 2000. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling, In *Proceedings of the SpaceOps 2000 Conference*, Toulouse, France.

Estlin, T., Gray, A., Mann, T., Rabideau, G., Castano, R., Chien, S. and Mjolsness, E. 1999. An Integrated System for Multi-Rover Scientific Exploration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 613-620. Orlando, FL.

Estlin, T., Rabideau, G., Mutz, D., and Chien, S. 2000. Using Continuous Planning Techniques to Coordinate Multiple Rovers. *Electronic Transactions on Artificial Intelligence* 4:45-57.

Joslin, D., and Clements., D. 1999. “Squeaky Wheel” Optimization. *Journal of Artificial Intelligence Research* 10:353-373.

Keeney, R. and Raiffa H. 1993. *Decisions with Multiple Objectives. Cambridge University Press.* New York, NY.

Minton, S., and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems.” *Artificial Intelligence*, 58:161-205.

Myers, K., and Lee, T. 1999. Generating Qualitatively Different Plans Through Metatheoretic Biases. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL.

Rabideau, G., Engelhardt, B., and Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO.

Williamson, M., and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL.

Zweben, M., Daun, B., Davis, E., and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.

Qualitative Decision-Theoretic Planning for Multi-Criteria Decision Quality

Abdel-Illah Mouaddib
GREYC-Université de Caen
Campus II - BP 5186
F-14032 Caen Cedex
mouaddib@info.unicaen.fr

Abstract

In this paper, we discuss the problem of optimizing the Multi-Criteria Decision Quality (MCDQ) under uncertain time constraint using Qualitative Markov Decision Process (Q-MDP). The main reason to use Q-MDP is that the multi-criteria quality is represented by a vector and it is hard to express the expected utilities, rewards and values with a numerical measure. The sequential decision multi-criteria quality process is modeled as an MDP where numeric reward and value functions are not always available. We describe different approaches using Q-MDP as an alternative of a classical MDP to deal with non-numeric reward and value functions. We also present an approach based on a numeric measure using an euclidean distance.

Introduction

In real-world applications, the solution quality is frequently a function of multiple criteria. Computational utility is represented, in general, as a single utility measure to computation based on the status of an n-tuple of criteria. One of applications concerned with such approach is the design of hybrid assembly lines, dealing with many criteria such cost, balance, reliability and congestion. The application of our concern is an exploratory rover that should visit different sites and develops experiments to collect information for scientists. At each site, experiments allow to collect information in return to the requests of scientists where each scientist is interested on particular information (chemical, geological analysis, etc ...). At each site, the rover takes pictures, makes analysis on stones, etc ... Tasks performed at each site improve the quality of information of each scientist. The decision-maker of the rover represents each scientist as a criterion. In order to maximize the satisfaction of all scientists (responding to the maximum number of requests), the rover have to decide the sequence of experiments to develop. This sequential decision process applied to an initial problem, yields a result having a quality described as a vector $\vec{Q}$ of quality criteria $(q_1, q_2, \ldots, q_n)$. The quality of a criterion in our context is the degree of satisfaction of a scientist (percentage of processed requests). Each decision (experiments at a site) improves the quality of the result according to a subset of criteria. The decision process is under time pressure because the response to the request for each scientist should be available before a fixed time. We use a class of resource-bounded reasoning based on progressive processing that uses a hierarchy of processing levels to solve a problem. This hierarchy allows to find a tradeoffs between the solution quality and computation time. When a progressive processing agent acts (executes a processing level), it improves some criteria of the quality. For the rover, the progressive processing task consists of a first processing level that allows to take pictures and analyzing them and a second level that makes experiments. The rover can decide to execute just the first level at a site. Abusively, we name an agent the execution of a progressive processing task at a site.

This application can be seen as a specific multiple objectives optimization problem that is also a particular application of MCDQ where each objective is a criterion of the overall solution. This problem has to address twin issues of searching in large and complex solution space and addressing, multiple, potentially conflicting objectives. Selection of a solution from a set of possible ones on the basis of several criteria is considered as a difficult problem. Due to this difficulty, most of researchers reduce the problem to a mono-criterion. Mathematical programming techniques and the popular weighted-sum approach have been developed (Yager 1988; Slany 1994). This problem has been extensively addressed by classical genetic algorithms using scalar fitness information. Other genetic algorithm use ranking methods to grade the population in terms of Pareto dominance. Most of those approaches are a kind of local optimality search rather than a global one.

In our context, to find the optimal multi-criteria quality consists in finding the optimal sequence of processing levels of multiple progressive processing agents. The decision process is sequential and consists in knowing which processing level should be executed next. This sequential decision process can be modeled by a stochastic automata with a Markov property. A Markov Decision Process (MDP) controller for pro-

gressive processing has been extensively discussed in (Mouaddib & Zilberstein 1998). However, the use of vectors to represent multi-criteria quality leads to new problems where numerical, additive utility, reward and value functions are not available. Indeed, if we assume that the vectorial space of multi-criteria quality is not Euclidean, a numerical definition of utility based on the Euclidean distance measure is not available. Thus, it is more appropriate to represent preference over states with an ordering relation rather than with additive utilities and rewards. A new interest in using a qualitative version of decision theory has been already advocated in (Dubois & Prade 1995; Boutilier 1994; Tan & Pearl 1994). This paper is another contribution in this line of research. Furthermore, it advances the state of the art in the control of resource-bounded agents by introducing the dependency and also in the MultiCriteria Decision by introducing the uncertainty.

Multi-Criteria Decision Quality Optimization problem

We use a vector of criterion quality represented by $(q_1^i, q_2^j, \ldots, q_n^k)$ where each criterion quality q_i^j represents the quality of the dimension i of the value in the solution quality after the execution of level l_i^j (j^{th} level of the agent i). Each criterion quality q_i^j is normalized ($q_i^j \in [0,1]$). Applications where the quality is not normalized we can use the percentage of improvement so that the measure belongs to [0,1]. In the context of the rover, we represent the quality at each dimension as the degree of satisfaction of the scientist (satisfied at 75% for example that can mean that 75% of scientist requests are satisfied). An agent i is the execution of a progressive processing task at a site i. Processing levels are taking pictures (first level) and analyzing stones (second level). We can also imagine further levels as analyzing pictures.

The vector $\vec{Q}$ represents a point in the vectorial space and each action that modifies one or many criteria moves this point in the space. The progressive processing is designed such that this point moves towards the absolute optimal point represented by the vector $\vec{1}$ (all the scientists are completely satisfied 100%). Indeed, we assume in the rover application that after each experiment the satisfaction of scientist increases. We discuss in the following if the trajectory of the point follows the Euclid assumption or not and describing the appropriate decision-theoretic approach.

Decision-Theoretic approach

Preliminaries

Performance profile Each individual agent i has a characterization of its performance that maps the status of an input quality vector $\vec{Q}$ to a discrete probability distribution of the duration t and output quality q_i^j that is the quality of dimension i in the vector after the execution of j^{th} level. This performance profile allows to express the probability to get a vector of satisfaction degrees of scientists given their current satisfaction degrees when we act at a special site.

The performance profile is denoted $PP_i((\vec{Q}', t)|\vec{Q})$. This performance profile is conditional as it is described in (Zilberstein & Russell 1993) and expresses the dependency between the status of the vector and the duration and the output quality of the processing level. The output quality can depend only on a subset of dimensions of the vector. This definition generalizes the definition of the conditional performance profile and allows to use a vector of dimensions to represent dependencies between each other. We discuss in the following the difficulties that can be raised to construct this performance and describing an alternative.

Time-dependent utility function The utility function $U(\vec{Q}, t)$ of the output vector quality represents the utility of the status of the quality vector at time t. This utility is multidimensional also and we represent it by the vector $(u_1(q_1, t), u_2(q_2, t), \ldots, u_n(q_n, t))$ where $u_i(q_i, t)$ is a numeric utility function of quality q_i at time t. The multi-dimensional utility allows us to express the utility of the satisfaction degree of a scientist after waiting t time units. We assume that when a scientist waits more than T time units the utility of the solution is null.

Formal framework of the MDP controller

The problem of the control consists in distributing available time T to progressive processing agents such that the multi-criteria decision is optimal given a probabilistic performance profile of agent processing levels and a utility time-dependency function. This problem can be seen as an MDP automaton where the states can be represented by the status of the vector when the time t is consumed. The rewards associated with a state are the utility of the state and the possible action is to execute the next processing level of the agent.

State representation The world of state is modeled as a stochastic automaton with a finite set of world states $\mathcal{S} = \{[\vec{Q}, t]\}$ where the *the initial state* is $[\vec{\mathbf{0}}, 0]$, the *terminal states* are $[\vec{Q}, T]$ and $[\vec{\mathbf{1}}, t]$ that represent respectively the state where the available time has been fully elapsed or the state where the highest vector quality is reached, and the *intermediate states* that code the status of the vector at time t. These states have the following form: $[\vec{Q}, t]$

Actions In every nonterminal state the possible action $\mathbf{E}_i^j$ is to execute the level j of the agent i in order to improve the quality of the criterion i (dimension i in the value of the result). There is quality dependency and uncertainty on execution time c_i^j (execution time of level j of agent i) and quality q_i.

Transitions The transition model is a function that maps each element of $\mathcal{S} \times \{\mathbf{E_i^j}\}$ into a discrete probability distribution over $\mathcal{S}$. In the following, the transitions

equations are described to code the transitions from an intermediate state to another and from an intermediate state to a terminal one.

Successful transition is when the action requires less resources than what is available. This transition is coded by:

$$if\ t + c_i^{j+1} \leq T,\ Pr([\vec{Q'}, t + c_i^{j+1}]|[\vec{Q}, t], E_i^{j+1}) \quad (1)$$

The vector $\vec{Q'}$ corresponds to the new status of the vector of quality after the execution of the level $j+1$ of the agent i that leads to improve the criterion i (quality stability assumption) of the vector quality $\vec{Q}$. Other criteria can be improved if we relax the stability assumption. Vectors $\vec{Q'}$ and $\vec{Q}$ have the following forms:
$\vec{Q} = (q_1^x, q_2^y, \ldots, q_z^a, \ldots, q_i^j, \ldots, q_n^k)$
• *Quality stability assumption*
$\vec{Q'} = (q_1^x, q_2^y, \ldots, q_z^a, \ldots q_i^{j+1}, \ldots, q_n^k)$
• *Quality stability assumption relaxed*
$\vec{Q'} = (q_1^x, q_2^y, \ldots, q_z^{a+1}, \ldots, q_i^{j+1}, \ldots, q_n^k)$
When the quality stability assumption is relaxed, vector $\vec{Q'}$ is the result of the modification of the qualities of more than one quality criterion of vector $\vec{Q}$. The rest of the discussion is not affected by the quality stability assumption.

Failure transition is when the action requires more resources than available. In such situations, the mechanism consists of aborting the action.

$$if\ t + c_i^k\ >\ T,\ Pr([\vec{Q}, T]|[\vec{Q}, t], E_i^j) \quad (2)$$

Expected Value

The value of intermediate states

$$V([\vec{Q}, t]) = \max_i (PP_i((\vec{Q'}, c_i^{k+1})|\vec{Q})V([\vec{Q'}, t + c_i^k]) + \sum_{\forall c_i^{k+1}\ c_i^{k+1}+t>T,\ \forall\ \vec{Q'}} PP_i((\vec{Q'}, c_i^{k+1})|\vec{Q})V([\vec{Q}, T])) \quad (3)$$

The value of terminal state

$$V([\vec{1}, t]) = U(\vec{1}, t) \quad (4)$$

$$V([\vec{Q}, T]) = U(\vec{Q}, T) \quad (5)$$

Theorem 1 *The optimal multi-criteria decision quality is the optimal policy computed for its corresponding MDP, given the utility function U.*

Proof:
Because of the one-to-one correspondence between the state of the MDP and the computation state of the multi-criteria decision quality problem, the optimal solution of the MDP is the optimal multi-criteria decison quality. □

The resulting MDP is a finite-horizon MDP with no cycles, because we move to the next level per agent, the transitions move "forward" with no loops. This MDP can be easily solved for a reasonable size (relatively large state spaces) using standard dynamic programming algorithms or search algorithm AO^*. However, those techniques can be used when we can tranform the vector of values to a single one. But in our context, this tranformation is not always possible. Consequently, the following issues have to be addressed.

- **Multi-dimensional utility** : The utility of multi-criteria quality can be represented as a multi-dimensional utility. The problem with such utilities is how to decide that a multi-dimensional utility dominates another since operator of comparison such as *Max* can be applied only to a single value. In this paper we discuss how to construct an optimal policy using a multi-dimensional utility.
- **Estimation of the performance profile** : The performance profile allows to characterize the behavior of each agent. This behavior is represented by the effect of an agent on a criterion given the status of the vector. The construction of such a characterization requires an important tool of sampling to estimate it. We discuss a possibilistic approach as an alternative to the stochastic one.

Before discussing those issues in the rest of the paper, we discuss the policy that should be followed when we have preferences on the criteria.

Considering preferences on criteria

We have discussed the problem of sequencing multiple progressive processing agents for multi-criteria decision quality, in a general case. However, in this section we discuss a specific policy for sequencing multiple progressive processing agents where criteria are ordered according to their preference. The preference definition we use in our context is given as follows:

Definition 1 *Criterion i is preferred over j $i \succ j$ holds when: $\tilde{V}([\vec{Q}, t]) > \tilde{V}([\vec{Q'}, t])$ such that $\forall \vec{Q}$ and $\vec{Q'}$, if $q_i > q'_i$ then we have,*

$$\forall t, t',\ \forall q_j, q'_j\ Pr([\vec{Q}, t])V([\vec{Q}, t]) > Pr([\vec{Q'}, t])V([\vec{Q'}, t'])$$

Intuitively, this definition is the Pareto-dominance of a criterion that states that as soon as the quality of the preferred criterion at a state s is higher than the quality of the preferred criterion at a state s', then s is preferred over s' (expected value is higher). The expected value at a state, in this definition, introduces the probability to be in a state, Pr([$\vec{Q}$,t]), that we show how it is computed in our context.

Theorem 2 *In situations where there is a preference function $\succ$ over criteria, the best policy is to allocate all the time to the most preferred criterion until its saturation (q=1), then the next most preferred and so on until the available time has been fully elapsed. This strategy is optimal.*

Proof:
We assume that $q_1 \succ q_2 \succ q_3 \ldots \succ q_n$ corresponding to the preference order on criteria $1, 2, \ldots, n$. Let π be the policy of the theorem.

Definition 2 *We say that a policy π is optimal if, $\forall$ π' and $[\vec{Q}, t]$ $V_\pi([\vec{Q}, t]) > V_{\pi'}([\vec{Q}, t])$.*

Let us assume that current state is $[\vec{Q}, t]$. We can then compute the probability to be in state $[\vec{Q'}, t + c_i^j]$ when moving from current state $[\vec{Q}, t]$ to state $[\vec{Q'}, t+c_i^j]$ as follows:

$$Pr([\vec{Q'}, t + c_i^j]) = Pr([\vec{Q'}, t + c_i^j]|[\vec{Q}, t], \mathsf{E}_i^j) \quad (6)$$

Because of the equations 1 and 2, the above equation becomes :

$$Pr([\vec{Q'}, \min(t + c_i^j, T)]) = \sum_{\forall(\vec{Q'}, c_i^j)} Pr((\vec{Q'}, c_i^j)|\vec{Q}) \quad (7)$$

Let us assume, for contradiction, that from state $[\vec{Q}, t]$ we use another policy π' rather than policy π of the theorem such as $\pi' \neq \pi$.

We execute agent j by using policy π'. Action a' is the action indicated by policy π' at current state $(\pi'([\vec{Q}, t]))$. The value with this transition is:

$$V_{\pi'}([\vec{Q}, t]) = \sum_{\forall(q_j, \delta)} PP_{a'}((\vec{Q''}, \delta)|\vec{Q}) V_{\pi'}([\vec{Q''}, \min(t+\delta, T)]) \quad (8)$$

The transition with policy π is that we execute agent i where action a is the one indicated by policy π at current state $(\pi([\vec{Q}, t]))$:

$$V_\pi([\vec{Q}, t]) = \sum_{\forall(\vec{Q'}, \delta)} PP_a((\vec{Q'}, \delta)|\vec{Q}) V_\pi([\vec{Q'}, \min(t+\delta, T)]) \quad (9)$$

Because criterion i is preferred to j and that in equation (9) the quality of this criterion is improved ($q_i' > q_i$) and that in equation (8) criterion i is not modified, we have: $q_i' > q_i''$. Moreover because of the definition 2 we can say that:

$$Pr([\vec{Q''}, \min(t + \delta, T)])V_{\pi'}([\vec{Q''}, \min(t + \delta, T)]) < Pr([\vec{Q'}, \min(t + \delta, T)])V_\pi([\vec{Q'}, \min(t + \delta, T)]) \quad (10)$$

Because of the equation 7 we have:

$$\sum_{\forall(\vec{Q''}, \delta)} Pr((\vec{Q''}, \delta)|\vec{Q}) V_{\pi'}([\vec{Q''}, \min(t + \delta, T)]) < \sum_{\forall(\vec{Q'}, \delta)} Pr((\vec{Q'}, \delta)|\vec{Q}) V_\pi([\vec{Q'}, \min(t + \delta, T)]) \quad (11)$$

Because of equations 8, 9 and 11 we can say that:

$$V_{\pi'}([\vec{Q}, t]) < V_\pi([\vec{Q}, t]) \quad (12)$$

The maximization of the equation (3) has not been respected. Contradiction. □

Approximating multi-dimensional time-dependent utility

In this section we introduce a numeric utility computed as an Euclidean distance between the current quality vector and the vector $\vec{\mathbf{1}}$ when the vectorial space is Euclidean. In the other case, we assign to states a qualitative utility degree. This qualitative degree is defined from the utilities of the criterion qualities. In the following, we describe pessimistic and optimistic qualitative utility functions, preference-based qualitative utility function and a possibilistic approach.

Euclidean distance measure

One way to define this utility is to take the Euclidean distance between vector $\vec{Q}$ and vector $\vec{\mathbf{1}}$ representing the maximal quality of the solution and the cost of the time t. Vector $\vec{\mathbf{1}}$ is n-tuplet (1,1,...,1) while vector $\vec{\mathbf{0}}$ is n-tuplet (0,0,...,0)

$$U(\vec{Q}, t) = -distance(\vec{Q}, \vec{\mathbf{1}}) - Cost(t) \quad (13)$$

The function *cost* is the cost incurred in the system when consuming t time units. Moreover, the distance function that define the euclidian measure is defined as follows:

$$distance(\vec{Q}, \vec{\mathbf{1}}) = \sqrt{\sum_i (1 - q_i)^2} \quad (14)$$

The distance between vector $\vec{Q}$ and vector $\vec{\mathbf{1}}$ is used to measure the rewarded value of the state by having a higher value when vector $\vec{Q}$ is close to vector $\vec{\mathbf{1}}$. In addition to the mathematicl convenience of the distance measure, the policy $\pi_{distance}$ using this measure, allows us to prefer a vector over another when it is close to vector $\vec{\mathbf{1}}$ representing the highest quality (qualities has been normalised).

Pessimistic vs Optimistic qualitative utility

We adapt the framework of the MDP defined previously to design a pessimistic qualitative MDP (Pes-$QMDP$) and an optimistic qualitative MDP (Opt-$QMDP$) using qualitative utility definitions. We define pessimistic and optimistic qualitative utility as follows :

Definition 3 *A pessimistic utility associated to a state $[\vec{Q}, t]$ can be defined by :*

$$u^{Pes}(\vec{Q}, t) = \min_i(U(\vec{Q}, t).u_i(q_i, t)) \quad (15)$$

Definition 4 *An optimistic utility associated to a state $[\vec{Q}, t]$ can be defined by :*

$$u^{Opt}(\vec{Q}, t) = \max_i(U(\vec{Q}, t).u_i(q_i, t)) \quad (16)$$

The pessimistic or optimistic utility replaces the numeric utility definition of equation 1 by :

$$R(U(\vec{Q}, t)) = u^{Pes}(\vec{Q}, t) \quad (17)$$

$$R(U(\vec{Q},t)) = u^{Opt}(\vec{Q},t) \quad (18)$$

Equations 4 and 5 are modified accordingly as follows

$$V(\vec{1},t) = u^{Pes}(\vec{1},t) \quad (19)$$

$$V(\vec{1},t) = u^{Opt}(\vec{1},t) \quad (20)$$

$$V(\vec{Q},T) = u^{Pes}(\vec{Q},T) \quad (21)$$

$$V(\vec{Q},T) = u^{Opt}(\vec{Q},T) \quad (22)$$

Equation 3 with u^{Pes} means maximizing the utility of the less satisfied scientist. However, equation 3 with u^{Opt} means maximizing the utility of the most satisfied scientist. This expression is less intuitive and the optimality with this measure u^{opt} is not obvious. However, with measure u^{Pes} allows an optimal policy as we claim in the following.

Claim 1 *The optimal multi-criteria decision quality is the optimal policy computed for its corresponding Pes-MDP using* u^{Pes}.

Proof:
The conditions of Theorem 1 remain valid, we can say that with pessimistic qualitative utility the policy obtained is optimal.□

Preference-based qualitative utility

In the pessimistic vs optimistic qualitative utility, we represent the multi-dimensional utility by a mono-dimensional utility and then we use a single measure as usually to compute the expected values of states. In many applications as multi-objectives applications a mono-dimensional utility is inappropriate. However, a preference on multi-dimensional utility exists. We discuss how we can use this preference to compute expected value of states (Tan & Pearl 1994) and the optimal policy for a preference-based qualitative MDP (P-$QMDP$).

The preference μ on multi-dimensional utility $U(\vec{Q},t) = (u_1(q_1,t), u_2(q_2,t), \ldots, u_n(q_n,t))$ can be defined by :

$$\mu : D_1 \times D_2 \times \ldots \times D_n \rightarrow scale$$

where $D_i \subset \Re$ is the definition domain of the utility $u_i(q_i,t)$ and scale = {1, 2, ... h}. This preference ranking μ allows to assign a preference rank that corresponds to an order-of-magnitude approximation of the utility associated with space $D_1 \times D_2 \times \ldots \times D_n$. The intended meaning of a ranking is that regions $R_i \subset D_1 \times D_2 \times \ldots \times D_n$ are ordered such that $R_1 \succ R_2 \succ \ldots \succ R_h$. This ranking allows to define a qualitative utility, representing a level of satisfactory, by :

Definition 5 *$\mu(U(\vec{Q},t)) < \mu(U(\vec{Q}',t))$ iff the following condition holds : $U(\vec{Q},t) \in R_i$, $U(\vec{Q}',t) \in R_j$ and $R_i \succ R_j$*

Example 1 *Let's assume that we have two scientists. $D_{i\in 1,2}$ = [0,1] and we can have the following regions : $R_1 = [0.95,1]\times[0.95,1]$, $R_2 = [0,0.95(\times[0.95,1]$, $R_3 = [0.95,1]\times[0,0.95($, such that $R_1 \succ R_2 \succ R_3$.*

We prefer that the first and second scientist be satisfied at least with a degree 0.95 rather than only one of them is satisfied at least with a degree 0.95 and we prefer that scientist two be preferred to the scientist one. Consequently : $\mu(R_1) = 1$, $\mu(R_2) = 2$ and $\mu(R_3) = 3$

Definition 6 *A qualitative utility $QU(\vec{Q},t)$ is defined by $R(\mu(U(\vec{Q},t)))$ Where R is defined as a reward function assigned to each scale such that $R(\mu_1) \geq R(\mu_2)$ when $\mu_1 \leq \mu_2$.*

The vector $\vec{Q}$ moves throughout regions from the less preferred to the most one (progressive processing assumption). Some adaptations of this framework are needed where regions are not naturally ordered.

In the same way, we use qualitative utility QU instead of the numeric utility of equation 1 and we modify equations 4 and 5 accordingly such that :

$$V(\vec{1},t) = QU(\vec{1},t) \quad (23)$$

$$V(\vec{Q},T) = QU(\vec{Q},T) \quad (24)$$

The expected value of terminal state we manipulate in this framework is multi-dimensional. The expression of equation 3 remains valid but since we manipulate a vector instead of a single numerical value the operator max is replaced by **Vmax** defined such that :

Definition 7
$\mathbf{Vmax}(V(\vec{Q},t), V(\vec{Q}',t)) = \max(QU(\vec{Q},t), QU(\vec{Q}',t))$

Equation 3 for multi-dimensional expected value is then defined as :

$$V([\vec{Q},t]) = \mathbf{Vmax}_i (PP_i((\vec{Q}', c_i^{k+1})|\vec{Q})V([\vec{Q}', t+c_i^k]) + \sum_{\forall c_i^{k+1},\ c_i^{k+1}+t>T,\ \forall\ \vec{Q}'} PP_i((\vec{Q}, c_i^{k+1})|\vec{Q})V([\vec{Q},T])) \quad (25)$$

Claim 2 *The optimal multi-criteria decision quality is the optimal policy computed for its corresponding MDP using multi-dimensional expected value and a qualitative utility definition.*

Proof:
Conditions of Theorem 1 remains also valid in this framework, then the policy obtained with this qualitative utility is optimal. □

Possibilistic approach (Π-MDP)

We discuss in this section the most general situation. The framework is when the probabilities are not available or hard to construct. Indeed, we use PP_a with multi-dimensional input and a multi-dimensional output that can be hard to construct. In this framework, we propose a possibilistic MDP (Π-MDP) as introduced

in (Sabbadin 2000). Instead of assuming available probabilities, we assume that we have for the uncertainty of the output vector when acting a possibility distribution π measuring to what extent an output vector $\vec{Q'}$ and a computation time c are plausible when an agent i acts in the state $\vec{Q}$ of the vector: $\pi_i((\vec{Q'}, c)|\vec{Q})$. In the same way, the consequences are ordered in terms of levels of satisfaction as introduced in the previous section by a qualitative utility function μ.

(Sabbadin 2000) proposed two equations for qualitative decision to obtain optimal strategies that we adapt to our former notation as follows :

$$V_{opt}[\vec{Q}, t] = \max_i \max_{\forall(\vec{Q'},c)} \min\{\pi_i((\vec{Q'}, c)|\vec{Q}), V_{opt}[\vec{Q'}, t+c]\} \quad (26)$$

$$V_{pes}[\vec{Q}, t] = \max_i \min_{\forall(\vec{Q'},c)} \max\{\pi_i((\vec{Q'}, c)|\vec{Q}), V_{pes}[\vec{Q'}, t+c]\} \quad (27)$$

V_{opt} measures to what extent there exists a satisfactory plausible consequence, while V_{pes} measures to what extent every plausible consequence is satisfactory. More details on this approach are given in (Sabbadin 2000).

Our claim in this section is just to show that when utility and probabilities in an MDP are hard to construct Π-MDP is an interesting alternative.

Concluding remarks

We present a new approach to construct an optimal multi-criteria decision quality by sequencing processing levels of multiple progressive processing agents. The optimal multi-criteria decision quality problem has been reformulated by an MDP, a vector of dimensions (the quality criterion) and we then show the construction of an optimal policy. We also show that when considering the criteria preferences the optimal policy consists of sequencing the processing levels in respect to the sequence of agents that is defined by the order of criteria preference. We also present how to address mutli-dimensional utility and value functions and how to estimate the performance profile. We show an adaptation of possibilistic MDP to our former and the optimal policy.

In the same way this approach advances the state of the art of the problem of controlling resource-bounded agents using dynamic programming (Mouaddib & Zilberstein 1998) since we introduce dependency between agents.

Future work will concern the application of this approach to handle multiple resources of the rover such as storage capacity and power.

References

Boutilier, G. 1994. Towards a logic for qualitative decision theory. In *KR*, 75–86.

Dubois, D., and Prade, H. 1995. Possibility theory as a basis for qualitative decision theory. In *IJCAI-95*, 1925–1930.

Mouaddib, A.-I., and Zilberstein, S. 1998. Optimal scheduling for dynamic progressive processing. In *ECAI-98*, 499–503.

Sabbadin, R. 2000. Empirical comparison of probabilistic and possibilistic markov decision process algorithms. In *ECAI*, 586–590.

Slany, W. 1994. Scheduling as a fuzzy multiple criteria optimization problem. *CD-Technical Report 94/62, technical university of Vienna.*

Tan, S.-W., and Pearl, J. 1994. Qualitative decision theory. In *AAAI-94*, 928–933.

Yager, R. R. 1988. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, Cybernetics* 18:183–190.

Zilberstein, S., and Russell, S. 1993. Anytime sensing, planning and action: A practical model for control robot. In *IJCAI-93*, 1402–1407.

Learning Single-Criteria Control Knowledge for Multi-Criteria Planning

Ricardo Aler **Daniel Borrajo**
Universidad Carlos III de Madrid
28911 Leganés (Madrid), España
aler@inf.uc3m.es, dborrajo@ia.uc3m.es
+34-1-624-9418, +34-1-6249459

Abstract

Planning for quality is currently receiving increasing attention by the researchers in this field. In the past, few researchers focused on finding good plans according to some criteria, mostly number of steps in the solution. The single criteria planning optimization task is very hard given that planning for satisfacing is not solved yet, though there are already some very impressive quantitative results that are very promising. When dealing with multiple criteria for planning optimization, the task gets harder. Our approach to solving this task has always been learning control knowledge for efficiently obtaining "good" plans. However, we have always focused on single criteria for optimization (number of steps, price, time, ...). In this paper, we present a solution for learning control knowledge for planning with multiple criteria. The solution consists on learning separate control knowledge for each criteria, and then merging the resulting control knowledge (rules). We present preliminary results that show that this approach yields better results than learning for the multiple criteria at once.

Introduction

Planning to generate *a* solution/plan has been largely studied in the literature. On the other hand, the search for *optimal* or *good* plans had been discarded in the past due to its complexity. Usually, plan quality was measured in terms of solution length (or, alternatively, makespan). Lately, a growing number of researchers have focused in trying to efficiently generate good solutions according to one criteria (Nareyek 2001). A common problem to all approaches has been the complexity of the task. One approach to decrease this complexity relies on learning control knowledge to gain both in efficiency (number of solved problems, time spent on finding a solution) and quality (finding good solutions according to a specific criteria) (Estlin & Mooney 1996; Iwamoto 1994; Pérez & Carbonell 1994; Ruby & Kibler 1992).

HAMLET is one such systems that learns control knowledge to guide *efficiently* PRODIGY4.0 (Veloso *et al.* 1995), a nonlinear planner, to *good* solutions (Borrajo & Veloso 1997; Borrajo, Vegas, & Veloso 2001). The learned control knowledge is represented as a set of control rules. We showed how it was possible to improve both efficiency and quality of plans. However, plan quality was measured in terms of only one criteria: solution length, price, time to execute the plan, etc. This is not enough in many real-world problems in which people is interested on obtaining a reduction in terms of several criteria such as cost, time, resources usage, etc. Also, in many cases, there might be several different criteria to be used in different situations, so planners should allow the user to select at any given moment a criteria, and learning systems should learn different control knowledge depending on those criteria.

In this paper, we present the use of HAMLET for the task of learning control knowledge to guide the planner to *better* solutions according to several user-defined metrics. We are using PRODIGY4.0 as the planner, since it allows the user to declare how to compute different criteria explicitly for each operator. The declarative representation of quality metrics allows reasoning with those metrics while planning using a branch-and-bound technique.

In order to learn control knowledge for several criteria, we thought of two approaches:

- learning control knowledge separately for each criteria, and then merging the resulting sets of control rules; or
- learning control knowledge for a combination of the criteria through the definition of a function of the individual criteria.

Here, we present results that compare both approaches and draws some preliminary conclusions from the experiments.

Section presents the base planner, PRODIGY4.0 with the extension to PRODIGY that allows the user to define quality criteria. It also describes how HAMLET learns control knowledge. Section shows the results of the experiments we performed, and Section draws some conclusions. Finally, Section relate our work to others.

Planning and learning systems

This section describes the planner we have used in this article (PRODIGY4.0) and the learning system used.

PRODIGY4.0

In this work, we have used a state space planner called PRODIGY4.0 (Veloso *et al.* 1995). PRODIGY4.0 is a nonlinear planning system that follows a means-ends analysis. The inputs to the problem solver algorithm are:

- Domain theory, $\mathcal{D}$ (or, for short, domain), that includes the set of operators specifying the task knowledge, the object hierarchy, and a set of quality criteria;
- Problem, specified in terms of an initial configuration of the world (initial state, $\mathcal{S}$) and a set of goals to be achieved ($\mathcal{G}$); and
- Control knowledge, $\mathcal{C}$, described as a set of control rules, that guides the decision-making process.

PRODIGY4.0's planning/reasoning cycle, involves several decision points:

- *select a goal* from the set of pending goals and subgoals;
- *choose an operator* to achieve a particular goal;
- *choose the bindings* to instantiate the chosen operator;
- *apply* an instantiated operator whose preconditions are satisfied or con-tinue *subgoaling* on another unsolved goal.

We refer the reader to (Veloso *et al.* 1995) for more details about PRODIGY4.0. In this paper it is enough to see the planner as a program with several decision points that can be guided by control knowledge (CK). If no CK is given, PRODIGY4.0 might make the wrong decisions at some points, requiring backtracking and reducing planning efficiency. Figure 1 shows an example of CK represented as a rule to determine when the operator `unload-airplane` must be selected. CK can be handed down by a programmer or learned automatically.

```
(control-rule select-operator-unload-airplane
     (if (and (current-goal (at <object> <airport>))
              (true-in-state (inside <object> <airplane>))
              (type-of-object <airplane> AIRPLANE)
              (type-of-object <airport> AIRPORT)))
     (then select operator unload-airplane))
```

Figure 1: Example of a control rule for selecting the unload-airplane operator.

QPRODIGY is an extension to PRODIGY in which knowledge about plan quality is encoded in the domain definition, i.e., in the set of operators (Borrajo, Vegas, & Veloso 2001). The user can define more than one quality function for each operator, as well as the metric used to measure the plan quality when solving the problems. When solving a problem using a given metric, the search for the first solution is common to that of PRODIGY. From that moment, QPRODIGY searches for *better* solutions by pruning all search paths that would lead to worse (more costly) solutions according to the metric (branch-and-bound approach).

In order to reason about quality, the domain definition is extended and operators are now defined by its preconditions, effects and cost functions. Each cost function may use numbers, domain variables, or, in general, any user-defined function for its computation. The only restriction for them is that they must return a numeric value: the cost of that particular metric. Every time an instantiated operator is applied, the cost of such operator is added to the cost of the applied operators belonging to the current search path. As an example, Figure 2 shows the representation of an operator in the Zenotravel domain used for the planning competition at AIPS'02. This operator allows an airplane to fly fast, taking less time than usual, but also consuming more fuel than usual.

```
(OPERATOR ZOOM (params <a> <c1> <c2>)
  (preconds
    ((<a> AIRCRAFT)
     (<c1> CITY)
     (<c2> (and CITY (diff <c1> <c2>)))
     (<d> (and DISTANCE
            (gen-from-pred (distance <c1> <c2> <d>))))
     (<b> (and BURN (gen-from-pred (fast-burn <a> <b>))))
     (<cf> (and CONSUMED-FUEL (consumed-fuel <d> <b>)))
     (<ca> (and CONSUMED-FUEL
             (gen-from-pred (capacity <a> <ca>))))
     (<f> (and CONSUMED-FUEL
            (gen-from-pred (fuel <a> <f>))
            (more-fuel-than-consumed <f> <cf> <ca>)))
     (<f1> (and CONSUMED-FUEL (new-fuel <f> <cf>))))
    (and (at <a> <c1>)
         (fuel <a> <f>)))
  (effects
    ((del (at <a> <c1>))
     (add (at <a> <c2>))
     (del (fuel <a> <f>))
     (add (fuel <a> <f1>))))
  (costs ((<speed>
            (and SPEED
                 (cost-from-pred (fast-speed <a> <speed>))))
          (<time>
            (and DURATION
                 (new-duration <d> <speed>))))
    ((TIME <time>)
     (FUEL <cf>)
     (TIME-FUEL (+ <time> <cf>)))))
```

Figure 2: Example of an operator in the Zenotravel domain for flying an airplane fast.

In order to use the cost functions, QPRODIGY has two new arguments with respect to PRODIGY4.0: the cost-function to plan for (by default, plan length), and the upper bound for the cost of any solution (by default, the cost of the first solution found, so the first solution produced by QPRODIGY will be the same as the one generated by PRODIGY). A standard branch-and-bound technique is then performed.

HAMLET

HAMLET is an incremental learning method based on EBL (Explanation Based Learning) and inductive refinement (Borrajo & Veloso 1997). The inputs to HAMLET are a task domain ($\mathcal{D}$), a set of training problems ($\mathcal{P}$), a quality measure (Q) and other learning-related parameters. A quality metric measures the quality of a plan in terms of number of operators in the plan, execution time, economic cost of the planning operators in the plan or any other user defined criteria. The output is a set of control rules ($\mathcal{C}$). HAMLET has two main modules: the Bounded Explanation module, and the Refinement module. Figure 3 shows HAMLET modules and their connection to PRODIGY4.0.

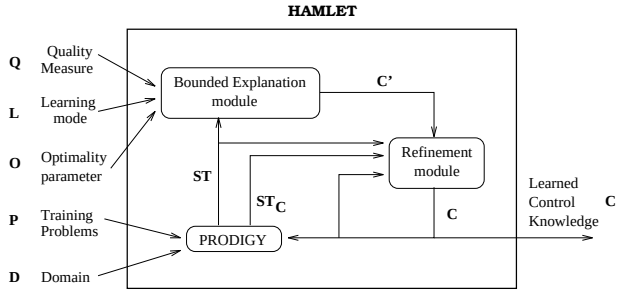


Figure 3: HAMLET's high level architecture.

The Bounded Explanation module generates control rules from a PRODIGY4.0 search tree. The details can be found in (Borrajo & Veloso 1997). The rules might be overly specific or overly general. HAMLET's Refinement module solves the problem of being overly specific by generalizing rules when analyzing positive examples. It also replaces overly general rules with more specific ones when it finds situations in which the learned rules lead to wrong decisions. HAMLET gradually learns and refines control rules, in an attempt to converge to a concise set of correct control rules (i.e., rules that are individually neither overly general, nor overly specific). ST and $ST_{\mathcal{C}}$ are planning search trees generated by two calls to PRODIGY4.0 planning algorithm with or without control rules, respectively. $\mathcal{C}$ is the set of control rules, and $\mathcal{C}'$ is the new set of control rules learned by the Bounded Explanation module.

Experiments and Results

The goal of the experiment has been to compare the two approaches for generating control knowledge for multiple criteria planning. We will use the quality-based planner, QPRODIGY, together with the learning system that we have discussed in the previous section, HAMLET. We chose the Zenotravel domain given that it has been used in the AIPS'02 planning competition. The Zenotravel domain is a version of the logistics domain in which several people have to travel from some initial cities to other ones. They can only use airplanes to move. Airplanes can fly using two different operators: `fly` (slow move, less fuel consumed); and `zoom` (fast move, more fuel consumed). When airplanes do not have more fuel they can refuel. The standard version of this domain provides three different quality criteria to be minimised: time, fuel, and time-fuel (addition of time and fuel criteria). The overall quality of a plan is computed as the sum of the costs of the individual operators in the plan.

We trained HAMLET with 200 problems of one goal to up to four persons to travel for each criteria. Separatedly, it generated 42 control rules for the time measure, 40 control rules using the fuel measure, and 48 control rules for the time-fuel measure. Then, we tested all configurations using 80 randomly generated test problems of 5, 10, 15 and 20 goals, a random number of persons to travel of upto 10, 15, 20, and 25. The results of the experiments are shown in Table 1. Values in the table have been accumulated for all the testing problems. The configurations named PRODIGY-*cost-function* were obtained by running QPRODIGY without control knowledge, using as a quality measure *cost-function*. Our experiments were performed in a 1'5 Ghz machine, with 0,5 Gb of RAM. We set a time bound of 10 seconds.

Table 1 shows the number of solved problems of each configuration in the Zenotravel domain, the total time, the length of the solution, and the quality of the solution according to each criteria. Given that all problems were solved by all configurations, we randomly generated 20 more difficult problems using 20 test problems of 50 goals, a random number of persons to travel upto 50. The results of the experiments are also shown in Table 1 in the rows starting with 20, only for the multiple criteria configurations (time-fuel). In order to compare configurations, it is enough to see the left hand side and the right hand side of the table. In order for the comparison to be fair, only the problems that have been solved by both systems are used to compute the accumulated time and quality values.

As an example of the control rules generated by HAMLET when learning for improving the cost-function `time`, Figure 4 shows a control rule that selects the operator `zoom`.[1] This control rule forces PRODIGY to use the operator that allows an airplane (<aircraft-235227-2>) to move fast from a city (<city-235227-4>) to another destination city (<city-235227-3>). This operator should be preferred to the other one for moving an airplane (`fly`) when the quality criteria is `time`. The

[1]Some irrelevant conditions in the `if`-part of the control rule have been removed for clarity, such as the type constraint of each variable.

Table 1: Results for several configurations in the AIPS'02 Zenotravel domain.

Number of problems	Solved problems	Time	Solution length	Solution quality	Solved problems	Time	Solution length	Solution quality
	PRODIGY-TIME				HAMLET-TIME			
80	80	4.0	1921	1,556,381	80	26.0	1856	1,241,064
20	20	1.0	370	71,740	9	20.0	372	68,519
	PRODIGY-FUEL				HAMLET-FUEL			
80	80	4.0	1921	215,895	80	12.0	1766	197,240
20	20	19.0	1875	158,006	20	64.0	1702	134,650
	PRODIGY-TIME-FUEL				HAMLET-TIME-FUEL			
80	80	4.0	1921	1,772,276	80	26.0	1821	1,554,373
20	20	3.0	592	211,642	11	38.0	534	183,505
	PRODIGY-TIME-FUEL				HAMLET-MERGING-TIME-FUEL			
80	80	4.0	1921	1,772,276	80	17.0	1766	1,398,419
20	20	10.0	1220	420,719	16	50.0	1105	349,154
	HAMLET-TIME-FUEL				HAMLET-MERGING-TIME-FUEL			
20	11	38.0	534	183,505	16	14.0	527	177,636

contrary is true when the other quality measure is selected, `fuel`.

```
(control-rule REDEDUCED-SELECT-ZOOM-TEMP-PROBLEMS-0-68-ENEIL-235227
 (if (current-goal (at <aircraft-235227-2> <city-235227-3>))
     (true-in-state (fuel <aircraft-235227-2> <infinite-235227-5>))
     (true-in-state (at <aircraft-235227-2> <city-235227-4>))
     (true-in-state (at <aircraft-235227-1> <city-235227-3>))
     (some-candidate-goals nil))
 (then select operators zoom))
```

Figure 4: Example of a control rule for selecting the `zoom` operator.

This control rule can be improved in two different ways: removing unnecessary conditions (the initial position of another airplane is not needed, <aircraft-235227-1>); and imposing another constraint on the fuel that the airplane has, so that it has enough for moving the airplane fast. The first possible improvement on the control rule can be obtained from either running new learning problems, allowing HAMLET to refine this rule, or using another learning system, such as EVOCK (Aler, Borrajo, & Isasi 1998) to get rid of those unnecessary conditions. However, the second improvement, imposing numerical constraints on variables, is much harder to obtain, since we would have to significantly modify the code for checking equality and subsumption of control rules when specializing or generalizing them.

Discussion and Conclusions

The most important result is that even though HAMLET is not prepared to handle numerical information, it is able to learn control rules that outperform PRODIGY4.0in terms of quality: plan length in all cases except for one (with a difference of only two operators in the plan) and fuel, time, and time-fuel in the respective configurations. On the other hand, it takes longer than PRODIGY4.0 to solve the problems with the learned knowledge (see the 'time' column[2]). This is reasonable because it takes time to evaluate the control rules, and also, it takes longer to solve a problem when quality is important than when it is not (it is well known that finding optimal solutions is a very hard problems, even in simple domains like the blocksworld). This is also probably the reason that some of the most difficult problems (the 20 problem set) are not always solved in the allocated time (this is so in the HAMLET-time, HAMLET-time-fuel, and HAMLET-merging).

There is another interesting result. Learning to solve each quality measure separately and then merging the control rules obtains better results than learning to

[2]The time to solve a problem should not be confused with the 'time' quality measure.

solve the combined time-fuel quality measure (1398 vs. 1554, smaller is better). We do not yet know why this is so, and how it depends on the function that combines the two quality measures (currently, the time-fuel measure is time+fuel). We will study this matter in the future.

Finally, it seems to be easier to decrease time than fuel because with the same computational effort, time is decreased 20.25% (315317 time units) and fuel is decreased only 8.64% (18655 fuel units). Decreasing time-fuel is between: 12.29%.

Related Work

There has been relatively very little work in the field of learning for obtaining good quality solutions as an explicit goal of the learning system. Moreover, very few have concentrated on the interaction between a human and a machine learning system for acquiring control knowledge for quality. However, there is an increasing interest now through the use of resources and time by the planners (Hasslum & Geffner 2001; Nareyek 2001; Ghallab & Laruelle 1994). Also, there is a strong relation to the whole field of scheduling, given that schedulers handle time and resources (Smith, Frank, & Jonsson 2000).

An earlier system, QUALITY system (Pérez & Carbonell 1994) also used the PRODIGY4.0 planner. While QPRODIGY defines a cost function for each operator, QUALITY needs as input an evaluation function of a whole plan. With respect to learning,

QUALITY compares the search trees produced by two different solutions to the same problem (one possibly generated by a human)[3] in order to learn control knowledge to *prefer* one solution to another. The problem with using *preference* control knowledge instead of using *selection* control knowledge (as is the case of HAMLET) is that when the planner has to backtrack, the alternatives are still there, and search can be much less efficient. An advantage of QUALITY is that it is able to transform the knowledge described in the evaluation functions into operative knowledge in terms of control knowledge. Another difference is that QUALITY does not refine the rules once they have been learned.

Other approaches define plan quality in terms of quality goals (Iwamoto 1994), carry out a rewriting process for optimizing the plan (Ambite & Knoblock 1998), or perform quality-based planning, without using learning as in PYRRHUS (Williamson & Hanks 1996). Within learning systems, others do not have the specific goal of improving solution quality; they obtain good solutions when they learn search control knowledge as a side effect. An example of such work are the first versions of the SCOPE system (Estlin & Mooney 1996), that uses a variation of FOIL (Quinlan 1990) to learn control knowledge for UCPOP (Penberthy & Weld 1992). They bounded the set of conditions to add to a control rule by using the information from the search trees. A newer version of this system was also able to obtain good solutions by learning, but they used only the "solution length" as their quality metric (Estlin & Mooney 1997).

Others employ a different strategy for learning, such as STEPPINGSTONE (Ruby & Kibler 1992), that learns cases for achieving good solutions, or reinforcement learning systems that acquire numerical information about the expected values of applying actions to states (Watkins & Dayan 1992). Reinforcement handles planning in a different way, since usually there is no explicit/declarative representation of operators. Learning relates to modifying numerical quantities associated to expected values of applying actions to states.

There has also been some work on planning using predefined notions of quality, such as PYRRHUS (Williamson & Hanks 1996), where optimal solutions were found by a version of the branch-and-bound technique, but there was no learning involved.

The work reported in (Ambite & Knoblock 1998) describes the planning by rewriting approach that allows to optimize solutions after a basic planning process has ended. In this case, instead of learning control knowledge, they allow the user to specify a set of rewriting rules for optimizing a generated plan. But, there is no learning, so it would be equivalent to allowing the user to define his/her own control rules.

[3]Therefore, it is difficult to compare its performance against HAMLET.

Acknowledgements

This work was partially supported by a grant from the Ministerio de Ciencia y Tecnología through project TAP1999-0535-C02-02.

References

Aler, R.; Borrajo, D.; and Isasi, P. 1998. Genetic programming and deductive-inductive learning: A multistrategy approach. In Shavlik, J., ed., *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, 10–18.

Ambite, J. L., and Knoblock, C. A. 1998. Flexible and scalable query planning in distributed and heterogeneous environments. In Simmons, R.; Veloso, M.; and Smith, S., eds., *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, 3–10. Pittsburgh, PA: AAAI Press.

Borrajo, D., and Veloso, M. 1997. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning* 11(1-5):371–405. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.

Borrajo, D.; Vegas, S.; and Veloso, M. 2001. Quality-based learning for planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources*, 9–17. Seattle, WA (USA): IJCAI Press.

Estlin, T. A., and Mooney, R. J. 1996. Multi-strategy learning of search control for partial-order planning.

In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume I, 843–848. Portland, Oregon: AAAI Press/MIT Press.

Estlin, T. A., and Mooney, R. J. 1997. Learning to improve both efficiency and quality of planning. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1227–1232. Morgan Kaufmann.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the 2nd International Conference on AI Planning Systems*.

Hasslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta, A., and Borrajo, D., eds., *Preprints of the Sixth European Conference on Planning (ECP'01)*, 121–132.

Iwamoto, M. 1994. A planner with quality goal and its speed-up learning for optimization problem. In Hammond, K., ed., *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS94)*, 281–286.

Nareyek, A., ed. 2001. *Working notes of the IJCAI'01 Workshop on Planning with Resources*. Seattle, WA (USA): IJCAI Press.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, 103–114.

Pérez, M. A., and Carbonell, J. G. 1994. Control knowledge to improve plan quality. In *Proceedings of the Second International Conference on AI Planning Systems*, 323–328. Chicago, IL: AAAI Press, CA.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Ruby, D., and Kibler, D. 1992. Learning episodes for optimization. In *Proceedings of the Ninth International Conference on Machine Learning*, 379–384. Aberdeen, Scotland: Morgan Kaufmann.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Watkins, C. J. C. H., and Dayan, P. 1992. Technical note: Q-learning. *Machine Learning* 8(3/4):279–292.

Williamson, M., and Hanks, S. 1996. Flaw selection strategies for value-directed planning. In Drabble, B., ed., *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS96)*, 237–244.

Why is difficult to make decisions under multiple criteria?

F. Della Croce
Dipartimento Automatica Informatica
Politecnico di Torino
dellacroce@polito.it

A. Tsoukiàs,
LAMSADE - CNRS
Université Paris Dauphine
tsoukias@lamsade.dauphine.fr

P. Moraïtis
Dept. of Computer Science
University of Cyprus
moraitis@ucy.ac.cy

Abstract

The paper makes a survey of the principal difficulties the multiple criteria decision making introduces with a particular emphasis on scheduling problems. Two types of difficulties are considered. The first is of conceptual nature and has to do with the difficulty of defining the concept of optimality in presence of multiple criteria and the impossibility to define universal preference aggregation procedures. The second difficulty is of more technical nature and concerns the increasing computational complexity of multiple criteria decision making problems. A number of examples are introduced in order to explain these issues.

Introduction

In this paper, decision making is referred to an agent (artificial or human) who has to act within a given context, with a given amount of resources and time in order to pursue one or more goals. The decision process is expected to be characterised by a form of rationality (possibly bounded) and to be represented in a formal way (the agent has preferences expressed either under a value function or more simply as a binary relation on the set of consequences of his/her actions). This is the frame of operational research and/or decision theory, possibly under Simon's (Simon 1979) bounded rationality variant.

In real life, making decisions under multiple criteria is the standard situation: there are always different consequences to consider, there always more objectives and goals to satisfy, there are always more opinions to take in account. Under this point of view the presence of multiple criteria it should be considered the general case, while single criterion optimisation should be considered as a special case. This is not what happened in the history of OR, where the first contributions on the use of multiple criteria appeared in the late 60s, early 70s (Roy 1968; Geoffrion 1968; Zeleny 1974; Keeney & Raiffa 1976).

The difficulty to make decisions under multiple criteria is twofold. The principal difficulty is conceptual. OR and decision theory are based on the idea of a rational decision process represented by a single objective function to optimise. Such an idea simply does not apply in the presence of multiple criteria. Further on, some other conceptual difficulties arise. Is it possible to substitute optimality with another concept? Are there universal procedures solving multiple criteria decision making problems? We explore these issues in section 2. The second difficulty is more technical and has to do with complexity. We confine ourselves in scheduling problems in order to show that the presence of multiple criteria normally implies the increase of computational complexity of the problem also in apparently "easy" problems. We discuss this problem in section 3.

The paper is based on results which are well known in literature. The aim of the paper is to put together such results for a community such as the A.I. planning and scheduling one. Further on, we want to show the importance of an autonomous theory concerning decision making and support in presence of multiple criteria and the difficulties such an effort has to face.

The vanishing optimum

Can the concept of optimum vanish (Schärlig 1996)? Traditionally when we think about decision theory we think about optimisation: find the one best solution. From a strict mathematical point of view this is straightforward. Express your problem as a function F of your decision variables $x_1, \cdots, x_n$ and then find the minimum (or maximum) of the function. This is well defined since

$$\min(F(x_1 \cdots, x_n)) \Leftrightarrow F'(x_1 \cdots, x_n) = 0$$

where F' is the "derivate" of function F. But then, as soon as we consider more than one criteria (more objective functions) we have a set of functions $F_i, i = 1 \cdots, m$ and we should look for a solution X such that $\forall i \ F_i'(X) = 0$ and this is a problem since $\forall i \ F_i'(X) = 0$ can be an inconsistent sentence.

Example 0.1 *Consider two objective function F_1, F_2, both to minimise, such that $\min(F_1) = A = \max(F_2)$ and $\min(F_2) = B = \max(F_1)$. Clearly the sentence $\forall i \ F_i'(X) = 0$ is inconsistent.*

There is no way to guarantee that in presence of multiple criteria there exist feasible solutions such that all objective functions can be simultaneously optimised. What we learn from that?

Difficulty 0.1 *Unlike traditional optimisation, the presence of multiple criteria does not allow to establish an "objective" definition of "optimal solution".*

In other terms when we work using multiple criteria there is no mathematical definition of the solution. We have to introduce alternative concepts, less easy to define and moreover subjectively established. What are we allowed to establish in the frame of multiple criteria?

There is a set of feasible solutions which are the "natural" candidates for solving a multiple criteria decision making problem. These are the so-called Pareto solutions (or efficient solutions or non dominated solutions). We introduce the following notation:

$$\forall X, Y \; D(X,Y) \Leftrightarrow \forall i \; F_i(X) \leq F_i(Y) \wedge \exists k \; F_k(X) < F_k(Y)$$

We read: *solution X dominates solution Y, iff for all criteria X is at least as good as Y and there is at least one criterion where X is strictly better than Y.* It is clear that all feasible solutions which are **not** dominated are potentially solutions of our problem (a dominated solution is obviously not interesting). The problem is that the set of Pareto solutions can be extremely large (sometimes equal to the set of feasible solutions).

Example 0.2 *Consider three candidates A, B, C such that for criterion: 1 $A > B > C$, for criterion 2: $B > C > A$ and for criterion 3: $C > A > B$ ($>$ representing a preference). All three candidates are non dominated.*

What can we do? Roughly there are two ways to face the problem:
1. fix a function $\mathcal{F}(F_1, \cdots, F_m)$ and then try to optimise $\mathcal{F}$ (that is re-conduct the problem to a single criterion optimisation problem);
2. explore the feasible or the efficient set using a majority rule as this is conceivable in various voting procedures (that is, choose the Pareto solution preferred by the "majority" of criteria).

One single function

The basic idea is simple. Put together the different functions in such a way that we obtain one single value for each feasible solution. After all this is exactly what happens in all schools, university degrees, multi-dimensional indices, cost benefit analysis and hundred other examples of "more or less" simple aggregation functions where values expressed on different attributes are merged in one single value.

The interested reader can look in (Bouyssou *et al.* 2000) for a nice presentation of all the drawbacks and unexpected consequences of such an approach. We try to summarise.

- Such a global function does not always exist. To say it in other terms, the conditions under which such a function exist are not always possible to fulfill. First of all evaluation on the different objective functions have to commensurable. Provided it is the case, then it should be possible to compensate the values of one function with the values of another function. If this is possible then each subset of functions should be preferentially independent with respect to its complement (see (Keeney & Raiffa 1976) for a detailed presentation of this approach). Last, but not least, it is possible that the effort to adapt the information to these conditions results in a model which has nothing to do with the original problem.
- Fulfilling the conditions can be possible in principle, but impossible in practice. In the sense that the cost of obtaining the extra information (such as the trade-offs among the criteria, the trial-error protocol used in order to calibrate the global function etc.) can be simply to large with respect to the problem or even unattainable (see (Hobbs 1986; Svenson 1996; Mongin 2000) for a discussion on this issue, including the cognitive effort required for such an approach).
- In any case, even if such a function can be defined, further information is required in order to establish it. Such information concerns two non exclusive issues:
  - further preferential information (trade offs among criteria, ideal points in the criteria space etc.);
  - shape of the global function (additive, distance, non linear etc.).

  In human decision support usually is the client (or decision maker) who provides such information through a protocol of information exchange with the analyst. However, there is always some arbitrariness in this process since this information depends also on technical choices (for instance trade offs are necessary in an additive function, but not in the frame of scalarising constants; the reader can see (Steuer 1986; Vanderpooten 1989; Korhonen, Moskowitz, & Wallenius 1992) for more details).

  The problem is more difficult in the case of "automatic" decision support as with artificial agents. Either such an agent has to carry enough preferential information or it has to be able to support a dialog with a human providing such information. Moreover the agent should be aware of the technical knowledge necessary to define the global function. It is always possible to fix the global function (at least the shape) from the beginning, but then we impose a severe limitation to the agent's autonomy.

Let the criteria vote

Another option is to make the criteria vote as if they were parties in a parliament. The idea is simple. Given any two feasible solutions X and Y, X is better than Y if it is the case for the majority of criteria. Hundreds of parliaments, committees, boards, assemblies, use this principle of democracy.

The interested reader can again refer to (Bouyssou *et al.* 2000) for a critical presentation of the drawbacks and counterintuitive results such an approach presents. Again we summarise.

- There is no universal voting procedure. Since the 18th century we know that voting procedures are either manipulable (to some extend a minority can impose its will) or potentially ineffective (unable to find a solution) as can be seen in the following example (borrowed from (French 1988)).

Example 0.3 *Consider four candidates (A,B,C,D) and seven examiners (a,b,c,d,e,f,g). Each examiner gives a preference in decreasing order (1 is the best, 2 is the second best etc.). The following table is provided.*

	a	*b*	*c*	*d*	*e*	*f*	*g*
A	*1*	*2*	*4*	*1*	*2*	*4*	*1*
B	*2*	*3*	*1*	*2*	*3*	*1*	*2*
C	*3*	*1*	*3*	*3*	*1*	*2*	*3*
D	*4*	*4*	*2*	*4*	*4*	*3*	*4*

If we sum the ranks of each candidate we obtain $\sigma(A) = 15$, $\sigma(B) = 14$, $\sigma(C) = 16$, $\sigma(D) = 25$ *and clearly B is the winner. Suppose now that for some reason the candidate D could not participate to the selection. Being the worst one should expect that nothing changes. Unfortunately it is not the case. Recomputing the sum of the ranks we obtain* $\sigma'(A) = 13$, $\sigma'(B) = 14$, $\sigma'(c) = 15$ *and now A is the winner. This is tricky. On the other hand if we look on pure majorities we get that* $A > B$ *(five examiners prefer A to B),* $B > C$ *(five examiners prefer B to C) and* $C > A$ *(four examiners prefer C to A). There is no solution.*

Arrow (Arrow 1963) definitely solved the problem proving the following theorem.

Theorem 0.1 *When the number of candidates is at least 3, there exists no aggregation method satisfying simultaneously the properties of universal domain, unanimity, independence and non-dictatorship.*

where:
- universal domain means that there is no restriction on the preferences to aggregate;
- unanimity means that an aggregation procedure should not violate the unanimity;
- independence means that in order to establish if X is better than Y we consider only information concerning X and Y and nothing else;
- non-dictatorship means that there is no preference information which is more importante than others, such to impose its will.

The reader can see that although the conditions imposed by Arrow are very "natural" they are inconsistent. In other terms: there is no universal preference aggregation procedure. Either we choose for guaranteing a result and we take the risk of favouring a minority or we impose the majority rule and we take the risk not to be able to decide. Decision efficiency and democracy are incompatible.

- Suppose a voting procedure has been chosen. If it is manipulable then one should obtain the information necessary to control possible counterintuitive results. If it is a majority rule then the outcome could be an intransitive and/or incomplete binary relation. In such a case further manipulation is necessary in order to obtain a final result.

  As for the previous approach such further information is usually provided by the client (the decision maker) through a precise dialog. A number of guidelines apply here (see (Bouyssou *et al.* 2000)), but no structured methodological knowledge is available up today. In the case of automatic decision making things become much more difficult since an artificial agent should be able to understand the difference among several voting schemes and procedures.

What did we learn from the above discussion?

Difficulty 0.2 *There is no way to establish an universal procedure for a multiple criteria decision making problem. Either further information has to be gathered or "extra-problem" procedures have to be adopted. Either the quality of the outcome can be poor (but we are sure to have an outcome) or we require a nice outcome knowing that it might be impossible to obtain it.*

The fact that we have such "negative" results should not induce the reader to consider that multiple criteria decision making problems are just a mess. In real world decision makers make every day sound decisions using multiple criteria. What we have to give up is the idea of **THE** solution of a multiple criteria decision making problem. We need to accept locally, bounded to the available information and resources, satisfying solutions.

There is still one more open question. Suppose that for a given problem we establish a model (and a concept of good or optimal solution). Suppose also that a precise procedure has been adopted in order to put together the preferences on the different criteria. How "complicated" is to reach a solution?

Complexity issues

Let us assume that a well defined multiple criteria optimisation model is available and, without loss of generality, let us consider scheduling problems. For a comprehensive analysis on multiple criteria scheduling we refer to (T'kindt & Billaut 2002). We will deal with the simplest scheduling environment, namely the static single machine environment. We use the notation given in (Chen & Bulfin 1993) that extends to multiple objective problems the so-called *three-field* $\alpha/\beta/\gamma$ classification of Lawler (Lawler *et al.* 1993).

Consider a set N of n jobs where each job j has a processing time p_j, a weight w_j and a due date d_j, respectively. Given a schedule, for each job j we denote with C_j its completion time, with $T_j = \max\{C_j - d_j, 0\}$ its tardiness. Also, let $T_{\max}$ denote the maximum tardiness of the schedule. Finally, let U_j denote the unit penalty for job j being tardy: namely, $U_j = 1$ if $T_j > 0$, else $U_j = 0$.

If we refer to mono-criterion problems, we already encounter all main classes of computational complexity (see (Garey & Johnson 1979) for details): for instance, the $1||\sum w_j C_j$, the $1||\sum U_j$ and the $1||T_{\max}$ are polynomially solvable, whereas the $1||\sum T_j$ is weakly NP-hard and the $1||\sum w_j U_j$ and the $1||\sum w_j T_j$ are strongly NP-hard.

Consider the simplest multiple criteria environment, namely the bi-criteria one and the two main general approaches indicated previously for putting together the two criteria (a specific case of the first approach is considered for presentation purposes):
(1) fix a function weighting the two criteria by means of

a lexicographic rule (one criterion is designated as primary and the other criterion is designated as secondary);
(2) generate the set of efficient solutions (to be then explored by some majority rule). Notice that an optimal solution of (1) always belongs to the set of efficient solutions described by (2).

In the three-field scheduling notation, γ denotes the performance measure. Let γ_1 and γ_2 be the two performance measures for the bi-criterion problem. Consider, now, the above general approaches with respect to single machine bi-criteria problems. In case (1), the objective γ_1 is lexicographically more important than objective γ_2 and the corresponding problem will be denoted as $1||(\gamma_2|\gamma_1)$. In case (2), where the set of non dominated solutions must be determined the corresponding problem will be denoted as $1||\gamma_1, \gamma_2$.

The following result proposed in (Chen & Bulfin 1993) links the complexity of a problem with single objective γ_1 to the complexity of bi-criteria problems involving objective γ_1.

Theorem 0.2 *If $1||\gamma_1$ is NP-hard, then $1||(\gamma_2|\gamma_1)$ and $1||\gamma_1, \gamma_2$ are NP-hard.*

Theorem 0.2 indicates that there is little hope to efficiently handle multiple criteria problems if any of the related mono-criterion problems is difficult.

There are actually a few special cases where the bi-criterion lexicographic problem is polynomially solvable when the secondary objective induces a mono-criterion NP-hard problem.

An example of this peculiar situation is given by the $1||(\sum T_j|\sum C_j)$ problem. The $1||\sum T_j$ problem is known to be NP-hard in the ordinary sense, whilst the $1||\sum C_j$ is known to be optimally solved in polynomial time by sequencing the jobs in nondecreasing order of their processing times, the so-called SPT rule. In the $1||(\sum T_j|\sum C_j)$ problem, in order to optimise the primary objective, the SPT rule must be respected. However there may be ties, namely jobs with identical processing times. Only for these jobs it is possible to optimise the secondary criterion. But this is equivalent to solve a special case of the $1||\sum T_j$ problem with all identical processing times, this latter problem being optimally solvable in polynomial time by sequencing the jobs in nondecreasing order of the due dates (the well known EDD rule). Hence, the $1||(\sum T_j|\sum C_j)$ problem is polynomially solvable.

Analogously there are a few special cases where the bi-criterion lexicographic problem is pseudo-polynomially solvable when the secondary objective induces a mono-criterion strongly NP-hard problem. An example is $1||(\sum w_jT_j|\sum w_jC_j)$ problem which is NP-hard in the ordinary sense though the $1||(\sum w_jT_j)$ problem is NP-hard in the strong sense. These are the only *relative good news* we have.

The following theorem also proposed in (Chen & Bulfin 1993) links the complexity of cases (1) and (2).

Theorem 0.3 *If $1||(\gamma_2|\gamma_1)$ is NP-hard, then $1||\gamma_1, \gamma_2$ is NP-hard.*

Theorem 0.3 indicates that case (2) is at least as difficult as case (1). Let then focus on bi-criteria problems handled by means of a lexicographic approach. We have here pretty bad results as bi-criteria problems involving polynomially solvable mono-criterion ones are often already NP-hard.

For instance, consider the $1||(\sum w_jC_j|T_{\max})$ problem. Both the $1||\sum w_jC_j$ problem and the $1||T_{\max}$ problem are polynomially solvable. The $1||(\sum w_jC_j|T_{\max}$ problem, however, is NP-hard in the strong sense as shown in (Hoogeveen 1992). This is due to the fact that the primary objective $T_{\max}$ induces a constraint in the secondary objective of the type $T_j \leq T_{\max}\ \forall j$, that can be written as $C_j \leq d_j + T_{\max}\ \forall j$. By introducing a deadline $\overline{d_j} = d_j + T_{\max}$, we obtain $C_j \leq \overline{d_j}\ \forall j$. Hence, the above $1||(\sum w_jC_j|T_{\max})$ problem is equivalent to the $1|\overline{d_j}|\sum w_jC_j$ problem which is known to be NP-hard in the strong sense.

What happens is that the lexicographic weighting of criteria (that we have seen to be generally easier than the generation of the efficient solutions) induces a further constraint (well defined as the primary objective is polynomially solvable) in the solutions space: this nearly always induces untractable bi-criteria problems that are polynomially solvable when only the secondary criterion is considered. This is what occurs in terms of pure computational complexity.

Also in practice, however, the structural properties of the problem defined on the secondary criterion tend to be destroyed when the primary objective is introduced as constraint.

An example of this is given by the $1||\sum(T_j|T_{\max})$ problem. By the same approach applied previously, this problem can be shown to be equivalent to the $1|\overline{d_j}|\sum T_j$ problem. But the presence of the deadlines kills the nice decomposition structure (leading to a pseudo-polynomial dynamic programming algorithm) of the $1||\sum T_j$ problem as shown in (R. Tadei). At the present state of the art the $1||\sum T_j|T_{\max})$ problem is open with respect to the weakly or strongly NP-hardness status.

What did we learn then in terms of complexity?

Difficulty 0.3 *Even when we deal with the easiest well defined multiple criteria problems, we immediately fall into NP-hard problems. There is very little hope to derive polynomial algorithms for multiple criteria problems whatever is the complexity status of the corresponding mono-criterion problems.*

So, also in terms of computational complexity, we face pretty negative results. Rather than being discouraged by this situation (as for NP-hard mono-criterion problems several high quality meta-heuristics exist for multiple objective problems), we need to precise very carefully the goals of our decision making: for instance, there is nonsense in searching for the complete set of efficient solutions if such set has huge cardinality.

As an example, consider problem $1||\sum w_jC_j, \sum h_jC_j$ where each job j has two weights (w_j and h_j). It is possible to derive the set of all efficient solutions by means of an ϵ-constraint approach and each solution can be computed in polynomial time. However the $1||\sum w_jC_j, \sum h_jC_j$ problem is NP-hard in the ordinary sense as the number of effi-

cient solutions may not be polynomially bounded as shown in (Hoogeveen 1992).

Conclusions

In this paper we analyse the conceptual and technical difficulties associated to decision making problems in presence of multiple criteria. Three difficulties are discussed:
- the impossibility to introduce an "objective" definition of solution;
- the impossibility to define "universal" preference aggregation procedures;
- the increasing computational complexity even when each single criterion corresponds to an "easy" problem.

Despite the apparent negative nature of the above results we claim that the development of precise preference aggregation procedures, of heuristics adapted to the presence of multiple criteria, allow for a given decision making problem to find satisfying solutions. What we should keep in mind is that:
- it makes no sense to look behind "optimality", in any way it might be defined;
- the method which is going to be used in order to solve a multiple criteria decision making problem is part of the model of the problem and is not defined externally.

References

Arrow, K.J. 1963. *Social choice and individual values*. Wiley, New York, 2nd edition.

Bouyssou, D.; Marchant, Th.; Pirlot, M.; Perny, P.; Tsoukiàs, A.; and Vincke, Ph. 2000. *Evaluation and decision models: a critical perspective*. Kluwer Academic, Dordrecht.

Chen, C.H., and Bulfin, R.L. 1993. Complexity of single machine, multi-criteria scheduling problems. *European Journal of Operational Research* 70:115–125.

French, S. 1988. *Decision Theory*. Ellis Horwood, Chichester.

Garey, M., and Johnson, D. 1979. *Computers and Intractability*. Freeman and Company, New York.

Geoffrion, A. 1968. Proper efficiency and the theory of vector optimisation. *Journal of Mathematical Analysis and Application* 22:618–630.

Hobbs, F. 1986. What can we learn from experiments in multiobjective decision analysis? *IEEE Transactions on Systems Man and Cybernetics* 16:384–394.

Hoogeveen, J.A. 1992. *Single machine bicriteria scheduling*. Ph.D. Dissertation, CWI, Amsterdam.

Keeney, R.L., and Raiffa, H. 1976. *Decisions with multiple objectives: Preferences and value tradeoffs*. Wiley, New York.

Korhonen, P.; Moskowitz, H.; and Wallenius, J. 1992. Multiple criteria decision support - a review. *European Journal of Operational Research* 63:361–375.

Lawler, E.L.; Lenstra, J.K.; Kan, A.H.G. R.; and Shmoys, D.B. 1993. Sequencing and scheduling: Algorithms and complexity. In Graves, S.C.; Kan, A.H.G. R.; and Zipkin, P., eds., *Handbooks in Operations Research and Management Science: Logistics of Production and inventory*. North-Holland, Amsterdam.

Mongin, Ph. 2000. Does optimisation implies rationality? *Synthese* 124:73–111.

R. Tadei, A. Grosso, F. D. C. Finding the pareto optima for the total and maximum tardiness single machine scheduling problem. *Discrete Applied mathematics* forthcoming.

Roy, B. 1968. Classement et choix en présence de points de vue multiples: Le méthode electre. *Revue Francaise d'Informatique et de Recherche Opérationnelle* 8:57–75.

Schärlig, A. 1996. The case of the vanishing optimum. *Journal of Multicriteria Decision Analysis* 5:160–164.

Simon, H.A. 1979. Rational decision making in business organizations. *American Economic Review* 69:493–513.

Steuer, R.E. 1986. *Multiple criteria optimisation: Theory, computation, and application*. Wiley, New York.

Svenson, O. 1996. Decision making and the search for fundamental psychological regularities: what can we learn from a process perspective? *Organisational Behaviour and Human Decision Processes* 65:252–267.

T'kindt, V., and Billaut, J.C. 2002. *Multicriteria Scheduling*. Springer Verlag, Berlin.

Vanderpooten, D. 1989. The interactive approach in mcda: a technical framework and some basic conceptions. *Mathematical and Computer Modelling* 12:1213–1220.

Zeleny, M. 1974. *Lineat Multiobjective programming*. LNEMS 95, Springer Verlag, Berlin.

The MO-GRT System: Heuristic Planning with Multiple Criteria

Ioannis Refanidis & Ioannis Vlahavas

Aristotle University, Dept. of Informatics
54006, Thessaloniki, Greece
{yrefanid, vlahavas}@csd.auth.gr

Abstract

This paper enhances the GRT planner, an efficient domain-independent heuristic state-space planner, with the ability to consider multiple criteria. The GRT heuristic is based on the estimation of the distances between each fact of a problem and the goals. The new planner, called MO-GRT, uses a weighted A* strategy and a multiobjective heuristic function, computed over a weighted hierarchy of user-defined criteria. Its computation is based on sets of non-dominated cost-vectors assigned to the problem facts, which estimate the total cost of achieving the facts from the goals, using alternative paths. Experiments show that a change in the criteria weights or scales affects both the quality of the resulting plan and the planning time.

Introduction

This paper presents MO-GRT, an extension of the domain independent heuristic planner GRT (Refanidis, and Vlahavas, 1999a; 2001) with the ability to take multiple criteria into account simultaneously. The word criterion refers to any type of measurable quantity, which is of interest in the solution plan. These criteria are provided by the user, along with the definition of the problem. The kind of problems that MO-GRT handles successfully are characterized by linear aggregation of the criteria values and by the ability to set bounds on them. Moreover, MO-GRT can handle both monotonic and non-monotonic criteria effectively.

The MO-GRT heuristic consists in assigning a set of cost-vectors to each fact, in a pre-processing phase. A cost-vector is an estimate of the total cost of achieving the fact by applying actions to the goal state, whereas its elements correspond to the various criteria. Different vectors for the same fact correspond to alternative ways of achieving the fact. Then, in the search phase, the states are evaluated using both the known accumulated value of the past plan, and the estimated value of the remaining plan based on the cost-vectors of the state's facts. The search-space is traversed using a weighted A* strategy, which enables the planner to exchange planning time and plan quality.

The multiobjective heuristic search paradigm has been introduced by Stewart and White (1991), who extended the typical A* algorithm in a vector-valued state space. Applications of the multiobjective framework in planning are also found in (Fujimura 96; Moraitis and Tsoukias 2000; Williamson, and Hanks 1994). However, these works assumed a given domain-dependent heuristic function. Besides, in most cases, an attempt was made to find all the solutions using exhaustive enumeration and evaluation of all states of the search space. Our approach is different in that it deals with the construction of a vector-valued heuristic function in a domain-independent way. Moreover, it supports the definition of preferences among the criteria, allowing tradeoffs to take place. Finally, the aim is not to find all solutions; it is to find the best compromise between the solution quality and the available planning time.

This paper is structured as follows: First the main concepts of the single-objective planning are introduced and they are extended in the multiobjective framework. Then, after a brief presentation of the GRT planner, the MO-GRT planner is presented in detail. Performance measurements in a logistics domain, where multiple criteria have been defined, give a first idea of the potentiality of the framework. Finally, some future directions are indicated.

The Multiobjective Planning Problem

In single objective planning, a cost $c(a)$ is assigned to each action a, denoting e.g. duration, resource consumption etc. Similarly, an overall cost can be assigned to a plan, which is defined as the sum of costs of its individual actions, i.e. $c((a_1, a_2, ..., a_M))=\sum_{i=1}^{M} c(a_i)$. In case that finding optimal plans is computationally too expensive, it may be acceptable to find a near optimal one. However, near-optimality is a subjective notion and cannot always be well defined.

In a multiobjective planning problem, a vector of costs $\boldsymbol{v}(a)$[1] is assigned to each action a. In this case, the overall cost of a plan $(a_1, a_2, ..., a_M)$ is also a vector $\boldsymbol{v}((a_1, a_2, ..., a_M)) = \sum_{i=1}^{M} \boldsymbol{v}(a_i)$. In order to compare plans, an evaluation function E has to be defined over the space of the cost-vectors.

Definition 1 (Plan comparison). A solution plan P_1 is considered better than a solution plan P_2 for a given

[1] Throughout the text bold typeface is used to denote vectors.

evaluation function E, iff $E(\boldsymbol{v}(P_1))<E(\boldsymbol{v}(P_2))$.

Note that we consider the lowest values of the evaluation function best. However, this commitment does not reduce the generality of the proposed method. In the following paragraphs some desired properties that may have an evaluation function are presented.

Definition 2 (Additive property). An evaluation function E satisfies the additive property, iff for any two cost-vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ the cost of their sum is equal to the sum of their costs, i.e.:

$$E(\boldsymbol{v}+\boldsymbol{u}) = E(\boldsymbol{v})+E(\boldsymbol{u}),\ \forall\ \boldsymbol{v}, \boldsymbol{u} \qquad (1)$$

The additive property is satisfied by linear-form functions, which do not have bounds in the valid values of their arguments.

Corollary 1. For two sets of cost-vectors **V** and **U** and for an evaluation function E that satisfies the additive property, the minimum cost among the sum of any pair of cost-vectors $\boldsymbol{v}\in\mathbf{V}$ and $\boldsymbol{u}\in\mathbf{U}$ is equal to the sum of the minimum costs of **V**'s and **U**'s cost-vectors, i.e.:

$$\min_{v\in V,u\in U}(E(\boldsymbol{v})+E(\boldsymbol{u}))=\min_{v\in V}(E(\boldsymbol{v}))+\min_{u\in U}(E(\boldsymbol{u})) \qquad (2)$$

In case of an evaluation function satisfying the additive property, it is possible to adopt a single-objective approach to solve the planning problem, just by assigning the value $E(\boldsymbol{v}(a))$ to each action a. However, this approach cannot be adopted in case the evaluation function does not satisfy the additive property, which is usual. This is illustrated in Figure 1.

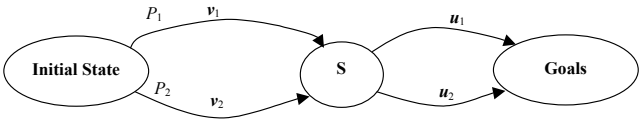


Figure 1: Evaluating a state in the presence of many criteria.

Suppose there is a state S of the state space, which can be reached by the initial state following two alternative paths, P_1 and P_2, with costs $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ respectively, so that $E(\boldsymbol{v}_1)<E(\boldsymbol{v}_2)$. In case of an additive evaluation function, path P_2 could safely be omitted, since, for any path leading from S to the goals with cost $\boldsymbol{u}$, $E(\boldsymbol{v}_1+\boldsymbol{u})<E(\boldsymbol{v}_2+\boldsymbol{u})$ (Corollary 1). Similarly, suppose there are two heuristic estimations $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ for the cost of reaching the goals from S, with $E(\boldsymbol{u}_1)<E(\boldsymbol{u}_2)$. In this case, the estimation $\boldsymbol{u}_2$ could be omitted, since, for any path P leading from the initial state to S with cost $\boldsymbol{v}$, $E(\boldsymbol{v}+\boldsymbol{u}_1)<E(\boldsymbol{v}+\boldsymbol{u}_2)$ (Corollary 1). The situation described above is identical with what is always the case in single-objective state-space planning, e.g. for each state, we need a single cost (and the corresponding path) to reach the state from the initial state and a single cost estimation to reach the goals from that state.

However, in case the evaluation function is not additive, we cannot omit neither the alternative paths between the initial state and any intermediate state nor the heuristic estimations of the cost of reaching the goals from any intermediate state. Thus, a large number of cost-vectors may have to be stored for each state, thus increasing the difficulty to solve a planning problem.

Another desirable property of an evaluation function is the monotonicity property. In order to define this property, we firstly introduce the domination relation between two cost-vectors $\boldsymbol{v}$ and $\boldsymbol{u}$, denoted with $\prec$.

Definition 3 (Domination). A cost-vector $\boldsymbol{v}$ is said to *dominate* another vector $\boldsymbol{u}$ and this is denoted by $\boldsymbol{v}\prec\boldsymbol{u}$, if for each i, $1\le i\le k$, v_i is better than u_i and $\boldsymbol{v}\neq\boldsymbol{u}$ (k stands for the vector dimensions).

The characterization "is better than" in Definition 3 may be interpreted either as "is lower than" or as "is higher than", depending on the nature of each dimension of the cost-vectors.

Definition 4 (Non-dominated vectors). A vector $\boldsymbol{v}$ is described as *non-dominated*, if there is no other vector dominating $\boldsymbol{v}$.

Having defined the concept of domination, we can introduce the monotonicity property for functions over vectors, extending the known monotonicity property of functions over real values.

Definition 5 (Monotonicity property). An evaluation function E is described as monotonic, iff $\forall\ \boldsymbol{v}, \boldsymbol{u}$: $\boldsymbol{v}\prec\boldsymbol{u}\Rightarrow E(\boldsymbol{v}) < E(\boldsymbol{u})$.

It can be proved that an evaluation function over a set of criteria satisfies the monotonicity property, iff its first derivatives on these criteria are continuously positive or negative functions, depending on whether the lowest or the highest values of the various criteria are considered best respectively.

The monotonicity property is satisfied by the evaluation functions of most real-world planning problems and helps to reduce the complexity of the planning process. This is achieved by leaving out all cost-vectors dominated by other vectors, thus keeping only the non-dominated ones. It is not difficult to show that all linear-form functions satisfy the monotonicity property.

The Single-Objective GRT Planner

The GRT planner is a domain-independent heuristic state-space planner (Refanidis, and Vlahavas, 1999a; 2001), which adopts the STRIPS formalism (Fikes, and Nilsson 1971). The term "domain-independent" refers to the way the heuristic function is constructed, i.e. a single algorithm is used for all domains. Its heuristic function estimates the distance, in terms of the number of actions, between any state and the goals, thus trying to minimize the plan length. However, the heuristic function is not admissible and GRT does not use an A* search strategy; instead, it either adopts the best-first search or the hill-climbing one. Thus, GRT, like all other effective heuristic planners, does not guarantee optimal plans.

The distance $dist(p)$ between each fact p and the goals is estimated in a pre-processing phase (heuristic construction

phase) by the following recursive formula:

$$dist(p) = \min \begin{cases} 0, \text{ if } p \in Goals. \\ \text{AGGREGATE}(Pre(a')) + 1, \text{ where } a' \text{ is an inverted action, such that } p \in Add(a'). \\ \infty, \text{ otherwise.} \end{cases} \quad \textbf{(3)}$$

In (3), the prefix operator *min* operates on sets of numbers and returns the minimum of them. The recursion follows from function AGGREGATE, which uses the distances of its arguments in order to produce its result, as it will be shown later in this section. Formula 3 is repeatedly applied until all distances stabilize. The distances obtained by (3) are used to further estimate the distance between each state of the state-space and the goals, by applying function AGGREGATE in the facts of each state, while searching in the space of the states.

A difficulty that may arise in the heuristic construction phase is that the actions of a problem cannot always be applied to the goals. GRT solves this problem by computing and using the inverted actions instead. For a normal actrion *a* than can be executed in state *s* and results in state *s*', the inverted action *a*' is an action that can be executed in *s*' resulting in *s*. Another difficulty that may arise is that in some planning problems, the goals do not constitute a complete state description, so it is not even possible to apply the inverted actions to them. GRT solves this problem by detecting the candidate missing goal facts and enhancing the goals with some or all of them, in a fully automated way (Refanidis, and Vlahavas 1999b; 2001).

In order to obtain more accurate and informative estimates, GRT introduces the notion of related facts. A fact q is considered related to another fact p, if the achievement of p leads to the achievement of q as well. The facts related to a specific fact p are called related facts of p and are denoted by $related(p)$. Intuitively, we can define the related facts of a set of facts P as the union of the related facts of P's facts, i.e. $related(P) = \bigcup_{p \in P} related(p)$. For an inverted action a' achieving a fact p, the related facts of p are defined by the following recursive formula:

$$related(p) = Pre(a') \cup related(Pre(a')) \cup Add(a') - Del(a') - \{p\} \quad \textbf{(4)}$$

which is initialized for the goal facts by setting $related(g) = \varnothing$, for each $g \in Goals$.

The related facts of a fact p depend on the specific path, i.e. the sequence of actions followed to achieve p. Since there are many paths to achieve a specific fact, there are many ways to define its related facts. For efficiency, GRT considers a single set of related facts corresponding to a path with minimum distance, for each fact. In case there are many alternative paths with the same minimum distance, GRT selects one of them arbitrarily.

Related facts play a critical role in function AGGREGATE. So, function AGGREGATE is a combination of sum and max functions, which groups its argument facts in disjoint sets of related facts and sums the maximum distances of each group. The full definition of the function AGGREGATE is the following:

Function AGGREGATE
Input: A set of facts $\{p_1, p_2, ..., p_N\}$, their distances $dist(p_i)$ and their lists of related facts $rel(p_i)$.
Output: An estimate of the cost of achieving the facts simultaneously.

1. Set $M_1 = \{p_1, p_2, ..., p_N\}$. Set $Cost = 0$.
2. While ($M_1 \neq \varnothing$) do:
   a. Let M_2 be the set of facts $p_i \in M_1$ that are not included in any list of related facts of another fact $p_j \in M_1$, without p_j being also included in their list of related facts. More formally:
   $M_2 = \{ p_i : p_i \in M_1, \forall p_j \in M_1, p_i \in rel(p_j) \Rightarrow p_j \in rel(p_i) \}$
   b. Let M_3 be the set of those facts of M_1 that are not included in M_2, but are included in at least one of the lists of related facts of the elements of M_2.
   $M_3 = \{ p_i : p_i \in M_1 - M_2, \exists p_j \in M_2, p_i \in rel(p_j) \}$
   c. Divide M_2 in disjoint groups of facts that are related to each other. For each group add the common cost of its facts to $Cost$.
   d. Set $M_1 = M_1 - M_2 - M_3$.
3. Return $Cost$

In (Refanidis, and Vlahavas 2001) it was shown that the set M_2 (step 2a of function AGGREGATE) will never be empty and it can always be partitioned in disjoint groups of facts achieved by the same action (step 2c). The number of iterations the function AGGREGATE performs is bounded by the initial size of M_1; however, one or two iterations are usually performed.

The Multiobjective GRT Planner

This section presents the MO-GRT planner in detail. The section starts with the definition of a criteria hierarchy, next presents the construction of the multiobjective heuristic function and finally presents how the states are evaluated using a weighted A*-like approach.

Evaluation Criteria

Plan evaluation criteria can be classified on the basis of several features. The first one refers to the values, higher or lower, that are considered best. We refer to the criteria of these two cases as *lower best* and *higher best* criteria.

Criteria can also be classified based on the direction in which their values are altered by the actions of a planning problem. The criteria, the values of which change in a single direction, are called *monotonic* (*increasing* or *decreasing*, based on the specific direction), while the

others are called *non-monotonic*. The monotonic criteria in particular can also be divided into *worsening monotonic criteria*, the values of which change towards their worst values, and into *improving monotonic criteria*, the values of which change towards their best values.

In decision making (Keeney, 1976; Vincke, 1992), criteria can be organized in hierarchies. For the lowest-level criteria, called *basic criteria*, a method of measurement is defined in order to assign values to them. For the highest-level criteria, called *compound criteria*, an aggregation method is defined, so that the values from the basic criteria can be combined and give an overall value for the evaluated object, i.e. the plan. MO-GRT adopts the Weighted Average Sum method (WAS), which is a linear multi-attribute value function, suitable for multi-attribute and multiobjective deterministic problems with arithmetic criteria and large numbers of evaluated entities, and results in a cardinal ranking among the alternatives. For the correct application of WAS, weights have to be assigned to the criteria, representing the relative preferences of the evaluator with respect to each criterion.

An example of a criteria hierarchy for a transportation logistics problem is shown in Figure 2. This hierarchy consists of two levels only; however, the basic criteria can be further analyzed to produce a deeper hierarchy.

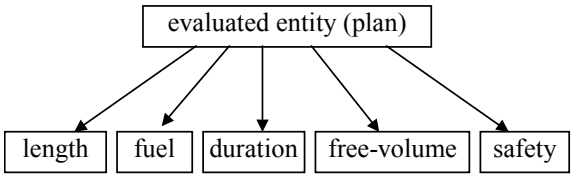


Figure 2: A simple criteria hierarchy for the logistics domain.

The criterion *length* is considered separately from the criterion of *duration*, since the former refers to the number of actions in a plan, while the latter refers to the cumulative duration of their sequential execution. Actually, the length of a plan reflects the difficulty in constructing it. Of course, in problems where all actions have equal durations, both criteria are equivalent and one of them should be omitted.

A scale is assigned to each basic criterion, including the indication whether higher or lower values are preferred. For example, the scale of *plan duration* for a specific problem may be the interval (20, 40) and lower values are preferred. This does not necessarily mean that plans with duration below 20 or above 40 time units will be pruned; it rather means that these plans will be evaluated as if they had a duration of 20 or 40 time units, respectively. The reason for setting scales for the basic criteria is twofold: Firstly, it prevents us from having extremely good plans, especially for the criteria the values of which can change towards their best values. Secondly, it allows us to normalize the values of all criteria in a common scale, in order to aggregate them.

Scales also play another role: Through the adoption of the WAS method for the aggregation of the values of the basic criteria, we implicitly considered that the evaluation function is linear. However, this assumption is too strong to be true in the whole real numbers interval. Thus, setting scales restricts this linearity within the scales only, which is a more actual assumption.

In many cases, the scale bounds are hard. For example, we might not accept a plan with duration greater than 40 time units. In the presence of hard bounds, MO-GRT prunes the plans that are definitely out of the bounds and gives low priority to plans estimated to be out of the bounds. In this paper, we use brackets to denote soft bounds and square brackets to denote hard bounds, e.g. (20, 40].

The definition of scales affects the results of the evaluation process significantly. For example, if all the produced plans are of a duration between 20 and 40 time units and we have set the scale of the criterion *duration* to the interval (0, 1000), then all plans will be considered as near optimal and will get about the same score with respect to *duration*. On the other hand, if we have set the scale of this criterion to the interval (20, 25), a plethora of plans with a duration of more than 25 time units will be considered as worst plans and will get exactly the same score. Deciding a criterion's scale is a critical issue and requires careful analysis of the problem and of the evaluator's preferences.

The Multiobjective Heuristic Function

The most difficult part of the evaluation process is the estimation of the cost of achieving the goals from each state of the state-space. MO-GRT extends the heuristic function of the single-objective GRT planner, by assigning each fact p cost-vectors of the form:

$$<Length, C_1, C_2, ..., C_N>$$

which estimate the cost of the various paths that achieve p from the goals (N stands for the number of basic criteria, whereas the criterion *length* is considered separately). The set of cost-vectors assigned to a fact p is denoted with $\mathbf{V}(p)$ and is computed by the following recursive formula, which generalizes Formula 3:

$$\mathbf{V}(p)=non\_dom \begin{cases} <0,0,...,0>, \text{ if } p \in Goals. \\ \textbf{AGGREGATE}(Pre(a')) + \\ \quad <1,r_1,...,r_N>, \text{ for each inverted} \\ \quad \text{action } a', \text{ so that } p \in Add(a'). \ r_i\text{'s}, \\ \quad i=1,..,N, \text{ denote the contribution of} \\ \quad a' \text{ to the basic criteria.} \\ <\infty, \infty, ..., \infty> \text{ otherwise.} \end{cases} \quad \textbf{(5)}$$

In (5), the prefix operator *non_dom* operates on sets of cost-vectors and returns the subset of non-dominated ones. Function **AGGREGATE** is identical with that of the single-objective GRT planner, except for the fact that it aggregates cost-vectors instead of single values.

If a set of cost-vectors is assigned to each fact p, then function **AGGREGATE** has to be applied to any combination of the different vectors of its arguments

resulting in a set of cost-vectors, i.e. a cost-vector for each different combination. Note that a different set of related facts is assigned to each cost-vector, depending on the specific sequence of actions that established this cost-vector.

Complexity Problems. MO-GRT faces the risk of combinatorial explosion both in memory and in time requirements. Memory requirements concern the space needed to store the non-dominated cost-vectors assigned to each fact. Even for two criteria only, the average number of cost-vectors per fact may be large. On the other hand, time requirements concern the application of function **AGGREGATE** to all combinations of the cost-vectors of its arguments.

Suppose $\overline{V}$ is the average number of cost-vectors per fact, $\overline{P}$ be the average number of precondition facts per action and $\overline{F}$ be the average number of facts per state. In this case, for the application of an inverted action in the heuristic construction phase, $\overline{V}^{\overline{P}}$ combinations should be considered on average. On the other hand, when estimating the cost of reaching the goals from a state of the search phase, only the best cost-vectors need to be considered first, according to the evaluation function. However, in case the resulting vector exceeds the hard bounds of some criteria, alternative combinations of the cost-vectors of the state's facts have to be considered. In the worst case, these combinations are $\overline{V}^{\overline{F}}$. For a better notion of these numbers, let us consider that $\overline{V}$=10, $\overline{P}$=3 and $\overline{F}$=30, which are some rather small values. In this case, we have an average of $\overline{V}^{\overline{P}}=10^3=125$ applications of function AGGREGATE for each applied inverted action during the heuristic construction phase and $\overline{V}^{\overline{F}}=10^{30}$ applications of function AGGREGATE in the worst case for each state during the search phase.

The Relaxed Dominance Pruning Heuristic. In order to overcome the complexity problems, MO-GRT adopts an alternative, more loose selection method in storing and combining cost-vectors. Henceforth, we refer to this method as the *relaxed dominance pruning heuristic* (RDPH). This method reduces the number of cost-vectors retained for each fact to the following ones:

- The best cost-vector, according to the criteria hierarchy.
- For each worsening monotonic criterion, the cost-vector with the best value in this criterion is also retained, regardless of whether the worst values are hard or soft bounded.
- For each improving monotonic criterion, a vector with the best combined value in the rest of the criteria, with respect to the vector of case 1, is also retained. However, in the rare case where the best value of such a criterion is hard bounded, the cost-vector with the worst value in this criterion is used instead.
- For the non-monotonic criteria, both case 2 and case 3 are applied, thus two additional cost-vectors are retained for each one of them.

The rationale underlying the selection of the above cost-vectors is the following: The best cost-vector of each fact is retained, since combining these cost-vectors of a set of facts (e.g. the preconditions of an inverted action or the facts of a state) may result to the best combined cost-vector for these facts, according to Corollary 1, which holds for the WAS function. However, this combined vector may probably exceed the scales for some criteria, so alternative vectors of the facts have to be tried. For the case worst bounds (either hard or soft) are violated, RDPH retains the vectors that have best values and are within the bounds (case 2).

The case where the best bounds are violated is treated separately, depending on whether these bounds are soft (which is the usual case) or hard. Violating the best soft bound of a criterion does not contribute positively to the overall value of the cost-vector, since the vector is evaluated as if it had the value of the bound. Thus, an attempt is made to find a new cost-vector that maximizes the overall value, even if it is still out of the bound for the specific criterion, rather than to find a new combined cost-vector inside the bound. This is the reason why in case 3 the vector that maximizes the combination of the other criteria is retained. However, in case the best bound is hard, then an attempt is made to produce a new cost-vector that falls inside the bounds; thus in this case, the vector with the worst value (due to the best strict bound) is also retained.

Suppose now that we have N basic criteria (*length* being excluded), N_1 of which are monotonic and N_2 of which are non-monotonic. In this case, the number of cost-vectors that will be retained for each fact would be $1+N_1+1+2\cdot N_2$.

In some cases, it is also possible to retain cost-vectors that exceed hard bounds of the scales. The strategy adopted is the following:

- The first cost-vector for each fact is retained, even if it exceeds some hard bounds.
- A new cost-vector that exceeds some hard bounds is rejected, provided that there is an existing cost-vector, which does not violate any hard bound to a greater extent than the new one.

Note that the criteria bounds used in the heuristic construction phase are not the original ones. Suppose a criterion c has a scale (L_c,R_c) and an initial amount of $Init_c$. In this case, the scale used for this criterion in the heuristic construction phase is (L_c-$Init_c$,R_c-$Init_c$). This is a consequence of the assignment of zero cost-vectors to the goal facts and this is because in the construction of the heuristic function we are only interested in the remaining cost of achieving the goals from any intermediate state and not in the cost paid for reaching the intermediate state from the initial one.

State Evaluation. In order to estimate the cost of

achieving the goals from any intermediate state, MO-GRT assigns a cost-vector to the state, by applying function AGGREGATE to the cost-vectors of the state facts. Certainly, there are many cost-vectors that can be produced, which represent the alternative paths in which the goals can be achieved from the current state. However, MO-GRT retains a single cost-vector only, which is considered to correspond to the "best" path.

Firstly, MO-GRT considers the best cost-vectors of the state-facts. In case the resulting vector does not violate any bound, it is assigned to the state. Note that, in this case, new scales reflecting the current resource availability are considered, rather than the original ones. Suppose the original scale of a criterion is (L_c, R_c) and the cost of the current plan with respect to this criterion is c. In this case, the scale of this criterion and for the specific state is considered to be (L_c - c , R_c - c).

In case the combined vector resulting from the best cost-vectors of the state facts violates some bounds, the alternative cost-vectors of the state-facts are attempted, giving priority to the vectors that reduce the extent of the violations and then to the vectors that improve the value of the resulting vector. As soon as a cost-vector that does not violate any bound is produced, the process stops and this vector is assigned to the state. However, in the worst case, this process would go on until all the combinations of the alternative cost-vectors of the state-facts have been considered without producing a non-violating combined cost-vector. In order to overcome the potential complexity problem in similar situations, MO-GRT reduces the number of alternative vectors tried in case of violations to the number of the criteria times the number of the initial violations.

Plan Evaluation

The criteria hierarchy is used to evaluate the states of the state-space. These must be evaluated both for the known accumulated cost of the past plan and the estimated cost of the remaining plan towards the goals, based on the heuristic estimations. Thus, the criteria hierarchy has to be applied twice and both values have to be combined. The only modification is that the criterion *length* is of no interest for the past plan, except for the case where this criterion reflects the duration of the plan.

The values assigned to the two top-level criteria, i.e. the past plan and the remaining plan, have to be combined using weights. The integrated function used for the evaluation of the states is formed as:

$$f(S)=W_p \cdot E(\boldsymbol{g}(S))+W_r \cdot E(\boldsymbol{v}(S)), \quad W_p+W_r=1,\ W_p \cdot W_r \geq 0 \qquad \textbf{(6)}$$

where S is the evaluated state, $\boldsymbol{g}(S)$ is the cost-vector of the past plan, $\boldsymbol{v}(S)$ is the cost-vector of the remaining plan that has been assigned to the state, W_p is the relative weight of the past plan and W_r is the relative weight of the remaining plan. For $W_p=W_r=0.5$, the search behaves as the original A* strategy, for $W_p=1$, the search behaves as a breadth first optimal strategy, whereas, for $W_r=1$, the search behaves as the greedy best-first strategy.

A crucial point is the treatment of the states, to which an estimated cost-vector of the remaining plan that violates some hard bounds has been assigned. MO-GRT cannot prune these states, since the MO-GRT heuristic is not admissible. So, the planner retains all states of the frontier set, however it penalizes the states that violate a hard bound by twice the amount of the violation.

This section ends with the application of the notion of domination to the states. The single-objective GRT keeps a closed list of visited states, in order to avoid re-visiting them. In the case of MO-GRT, this closed list has to be extended, in order to store the non-dominated cost-vectors of the several visits in the state. Now, a revisited state is only pruned in case the vector of a previous visit dominates the vector of the new one in all basic criteria.

Performance Measurements

This section examines the role of the criteria, their weights and scales play in the planning process, i.e. how they affect the planning time and the quality of the resulting plans. This is performed through an adequate number of experiments in an enhanced *logistics*-type domain.

The *logistics*^{MO} Domain

In the original logistics domain (Veloso 92), there is a single means of transportation to transfer an object between two cities, i.e. an airplane. In order to measure the effectiveness of MO-GRT, we have extended this description with trains, which can only perform transportations between different cities and we have labeled one location in each city as a railway station. We call this extended logistics domain *logistics*^{MO}.

The new domain has three new actions, referring to the loading, unloading and moving of a train. Moreover, two predicates, namely *train* and *station*, have been introduced: they describe an object as train and railway station, respectively. We have also introduced the criteria of *cost* and *duration* and we have assigned an application cost and duration to all domain actions schemas (Table 1). Certainly, lower values are preferable.

Table 1: Application cost and duration for the actions of the *logistics*^{MO} domain.

Actions	Cost	Duration
loading/unloading any truck	2	1
loading/unloading any train	2	1
loading/unloading any plane	3	2
moving a truck	10	10
moving a train	20	100
flying a plane	50	10

As it results from Table 1, all criteria (including *length*) are monotonically increasing. Besides, since lower values are preferred, the criteria are also worsening criteria. However, the adoption of a single type of criteria for the experiments does not restrict the generality of the results, since MO-GRT deals with all types of criteria equally.

Problem Definition

As a starting point, we used the STRIPS untyped *logistics* problem set of the AIPS-00 planning competition[1], comprising 28 problems. In every problem of this distribution, all cities have two locations, one of which is the airport. We labeled the non-airport location of each city as railway station. Furthermore, we added a train to each problem, initially located in the railway station of the first city. Note, finally, that the initial values of all criteria are considered to be zero.

To apply MO-GRT, we must set the criteria scales, which are not identical in all problems. In order to render the reproduction of the experiments feasible, we used an "algorithmic" way of setting these scales. Thus, these were based on the number of packages that had to be moved inside one city, or to a different one. We omitted packages that were not referenced within the goals, as well as packages with identical initial and goal positions. Table 2 shows the expressions used to set the scales for all criteria.

Table 2: Scales for the three criteria.
P_1=Packages that must be transferred inside the same city
P_2=Packages that must be transferred to a different city

Criteria	Left bound	Right bound
Length	(Right bound) / 4	$4 \cdot P_1 + 12 \cdot P_2$
Cost	(Right bound) / 8	$24 \cdot P_1 + 154 \cdot P_2$
Duration	(Right bound) / 8	$22 \cdot P_1 + 246 \cdot P_2$

The rationale of the above formulas is the following: As for the right bound, the formulas describe the worst cases, i.e. cases where the packages are transferred separately, while a means of transportation is never in place for transfer and it must be moved from another position. On the other hand, the decision for the left bound was based on our experience with the problems of the *logistics*^{MO} domain. The intention was to have all the solutions between the two bounds and to have a sensible distance between the left bound and the obtained values for all criteria.

We performed 12 experiments, denoted with the letters A to L. Each experiment included running the planner for all 28 problems of our *logistics*^{MO} problem set, using specific weights. Table 3 summarizes the weights used in these experiments.

Table 3: Weights used in various experiments.

	Weights				
Experiment	Past plan	Remaining Plan	Length	Cost	Duration
A	1	3	3	1	1
B	1	1	3	1	1
C	1	2	3	1	1
D	0	1	3	1	1
E	1	3	1	1	1
F	1	3	2	1	1
G	1	3	5	1	1
H	1	3	10	1	1
I	1	3	3	3	1
J	1	3	3	10	1
K	1	3	3	1	3
L	1	3	3	1	10

[1] http://www.cs.toronto.edu/aips2000/.

Note that in the *logistics*^{MO} domain, the criterion *length* does not clearly favor any of the other two criteria, since the average actions needed to perform a transportation are the same, whether a plane or a train is used. However, this would not be the case if, for example, loading a package in a plane would require more than one actions. In general, the criterion *length* is usually positively related to some criteria and negatively related to some others.

Experimental Results

MO-GRT has been implemented in C++. The measurements were taken on a SUN Enterprise 3000 machine running at 167MHz, with 256 MB main memory under Solaris 2.5.1 OS. We set a CPU time limit equal to 5 minutes for each problem. Some problems were not solved due to memory limitations or due to the requirement for more processing time.

Next we compare several groups of experiments, where each group includes experiments that only differ in a single weight. Experiment A is included in all groups and serves as a reference experiment. When comparing two experiments, e.g. experiment X to experiment A, the following metrics are used:

$$m_{solved} = \frac{solved(X) - solved(A)}{solved(A)}$$

$$m_{time} = \underset{i}{average}\left(\frac{time(X_i) - time(A_i)}{time(A_i)}\right)$$

$$m_{length} = \underset{i}{average}\left(\frac{length(X_i) - length(A_i)}{length(A_i)}\right) \qquad \textbf{(7)}$$

$$m_{cost} = \underset{i}{average}\left(\frac{cost(X_i) - cost(A_i)}{cost(A_i)}\right)$$

$$m_{duration} = \underset{i}{average}\left(\frac{duration(X_i) - duration(A_i)}{duration(A_i)}\right)$$

where:
solved(Z): the number of problems solved in experiment Z
time(Z_i):the time needed to solve problem i in experiment Z
length(Z_i):the length of the solution to the problem i in experiment Z
cost(Z_i): the cost of the solution to the problem i in experiment Z
duration(Z_i): the duration of the solution to the problem i in experiment Z
Z is an experiment (in this case X or A).

Note that the averages are computed in the problems solved in both experiments (X and A). There are two reasons why we use averages on a large number of problems, instead of comparing on specific problems. The first one concerns the large number of problems, the huge number of interesting variations in weights and scales, and

the five metrics of interest, which makes the detailed presentation of the results impossible in all cases, due to the limited space of the paper. The second reason is that the impact of changing a weight or a scale may be negligible or even contradictory for a specific problem, due to the heuristic nature of the proposed technique; however the average results are always intuitive.

Table 4 presents the absolute values in all 28 problems for experiment A, which are used as a basis of comparison in the subsequent sections. The problems have been solved with the relaxed dominance pruning heuristic.

Hereinafter we compare the performance of MO-GRT with and without the RDPH and then we examine the effect of the weights of the past and the remaining plans, as well as the weights and scales of all criteria, on the planning process. Finally, we compare MO-GRT to GRT in the same problems.

Table 4: Detailed results for experiment A (solution time in msecs).

Problem	time	length	cost	duration	Problem	time	length	cost	duration
logistics-4-0	170	13	126	238	logistics-9-0	540	26	296	480
logistics-4-1	180	15	162	152	logistics-9-1	410	22	186	170
logistics-4-2	130	12	164	158	logistics-10-0	930	29	308	490
logistics-5-0	190	19	170	156	logistics-10-1	1350	39	412	628
logistics-5-1	200	12	76	228	logistics-11-0	1190	36	440	618
logistics-5-2	120	8	32	26	logistics-11-1	1520	47	486	656
logistics-6-0	230	22	112	256	logistics-12-0	1120	35	326	502
logistics-6-1	130	9	52	126	logistics-12-1	1250	42	420	712
logistics-6-2	240	21	178	162	logistics-13-0	2530	53	648	774
logistics-6-9	170	15	132	242	logistics-13-1	2070	43	504	636
logistics-7-0	490	31	346	406	logistics-14-0	3330	47	422	832
logistics-7-1	550	33	308	406	logistics-14-1	2610	49	448	734
logistics-8-0	610	26	244	388	logistics-15-0	4510	64	686	724
logistics-8-1	800	29	284	384	logistics-15-1	3150	61	700	698

The Relaxed Dominance Pruning Heuristic. First, we investigate the effect of RDPH on the overall performance of the MO-GRT planner, using the weights and scales of experiment A. In the comparison we are interested in the metrics m_{solved}, m_{time} and m_{quality}. The last metric is defined by the following formula:

$$m_{quality} = \underset{i}{average}\left(\frac{quality(X_i) - quality(A_i)}{quality(A_i)}\right)$$

where $quality(Z_i)$ is the result of applying the WAS function on the solution plan of problem Z_i. This metric does not make sense in case different weights or scales are used; in such cases, we have different evaluation functions. On the other hand, since this section presents results on the overall quality of the resulting plans, we have omitted results for the individual metrics m_{length}, m_{cost} and m_{duration}. Table 5 presents the results.

Table 5: Performance of MO-GRT without RDPH.

Experiment	m_{solved}	m_{time}	m_{quality}
A (without RDPH)	-69.41%	865.37%	6.32%

The results presented in Table 5 can be interpreted as follows: Running MO-GRT in the 28 problems of our problem set, with the weights and scales of experiment A and without RDPH heuristic results in solving 69% less problems than when running MO-GRT with the RDPH in the same problems, with the same weights and scales and in the time limit of 5 minutes. Besides, in problems solved by both configurations of MO-GRT, the configuration without the RDPH needed 865% more time on average to solve them and produced plans that were 6.32% better on average than the configuration with the RDPH.

The above results are intuitive, since retaining all the non-dominated cost-vectors for the facts of a problem demands more processing time but gives us the opportunity to perform more effective tradeoffs, thus producing better plans. However, we believe that the slight increase in quality when retaining all non-dominated cost-vectors is unimportant compared with the significant degradation in the planner's efficiency. Thus, all measurements in the following sections have been taken with the RDPH, the default configuration of MO-GRT.

Weights of the Past and the Remaining Plan. In this section, we compare experiments B, C and D to experiment A. We investigate the effect of different combinations of the past and the remaining plans weights on the overall planning process. Actually, what is of interest is the ratio between these weights. Table 6 shows the results.

Table 6: Results for several combinations of past and remaining plan weights.

Experiment	m_{solved}	m_{time}	m_{length}	m_{cost}	m_{duration}
B (1/1)	-35.71%	913.6%	-1.01%	-0.88%	-8.68%
C (1/2)	0.00%	52.6%	-0.72%	-3.31%	-0.47%
D (0/1)	0.00%	-13.92%	0.99%	2.91%	4.75%

Table 6 shows that as the ratio between the weights of the past and the remaining plan decreases, the planner reaches faster a solution. Furthermore, in experiment B, where we had the highest value of this ratio, 35% of the problems (the largest ones) were not solved. With regard to the three criteria, as the above ratio increases, the produced plans generally become better. However, the degree of this effect is different for the three criteria.

Weight of the Criterion *length*. In this section, we compare experiments E, F, G and H to experiment A. We investigate the effect of the weight of the criterion *length* on the overall planning process. Table 7 shows the results.

Table 7: Results for several weights of the criterion *length*.

Experiment	m_{solved}	m_{time}	m_{length}	m_{cost}	m_{duration}
E (length=1)	0.00%	130.19%	3.76%	1.03%	-2.05%
F (length=2)	0.00%	14.32%	1.88%	-0.61%	1.14%
G (length=5)	0.00%	-7.38%	-0.49%	0.89%	-1.16%
H (length=10)	0.00%	-12.42%	-0.40%	1.29%	-1.09%

The above results show the effect of the weight of the criterion *length* on the total solution time. As the weight of this criterion increases, the planner reaches faster a solution and finds slightly shorter plans; on the other hand, as this weight decreases, the planner delays and finds slightly longer plans. The effect of this weight on the plan

cost and duration is neither significant nor consistent. This is explained by the fact that in the *logistics*^{MO} domain the plan length is not competitive either to the plan cost or the plan duration.

Weights of the Criteria *cost* and *duration.* In this section, we compare experiments I, J, K and L to experiment A. We investigate the effect of the weights of the criteria *cost* and *duration* on the overall planning process. Table 8 shows the results.

Table 8: Results for various weights of the criteria *cost* and *duration.*

Experiment	m_{solved}	m_{time}	m_{length}	m_{cost}	$m_{duration}$
I (cost=3)	0.00%	13.72%	-0.27%	-9.77%	7.06%
J (cost=10)	-17.86%	1509.18%	0.68%	-14.55%	9.16%
K(duration=3)	0.00%	64.74%	6.93%	9.89%	-9.26%
L (duration=10)	-21.43%	1161.74%	22.53%	32.29%	-17.72%

It is seen that for both the criterion *cost* (experiments I and J) and the criterion *duration* (experiments K and L), as the weight of each criterion increases, the resulting plans become better in terms of this criterion, while they worsen with respect to the rest of the criteria. It is also seen that an increase in the weight of the criterion *cost* does not significantly affect the length of the obtained plans, which does not occur in case of an increase in the weight of the criterion *duration*. The rationale of this observation is that most packages are initially located in railway stations and the same occurs in their goal positions. Thus, the demand for plans of lower duration favors the use of planes for transportation, which leads to longer plans as a side-effect.

A second observation is that as the weights of the criteria *cost* and *duration* increase, whereas the weight of the criterion of *length* remains the same, the solution time increases. Especially in case these weights become greater than the weight of the criterion *length* (experiments J and L), many problems cannot be solved within the time and memory limits. This result was expected, based on the previous results concerning the influence of varying the weight of the criterion *length*.

Scales of the Criteria. In this section, we investigate the effect of the scales attached to the criteria on the overall planning process. Reusing the problems of experiment A as a reference, we constructed 12 variations, keeping the same weights and changing the scales. The new experiments are denoted with $A_{criterion\ x\ M}$, where *criterion* is the criterion the scale of which has changed, and *M* is a positive number that multiplying the width of the original scale. The new scale has the same center as the original one, but it is *M* times broader. For example, if the scale of *duration* in a problem of experiment A was initially (200, 300) and *M* was 2, the same scale in experiment $A_{DURATIONx2}$ would be (150, 350). Both scales have the same center, i.e. 250, but the second one is two times broader than the first one. Note that in case the left bound becomes lower than 0, we set it to the value 0 and we shift the right bound accordingly. Table 9 shows the results.

The conclusion drawn from Table 9 is that a criterion scale affects significantly the quality of the resulting plan in terms of this criterion. The results show that as the scale of a criterion diversifies (broadens or shrinks) from a critical scale, the quality of the obtained plans reduces, in terms of this criterion, whereas it may increase in terms of the other criteria. For example, as we broaden or shrink the scale of the criterion *cost*, the cost of the obtained plans increases, while their duration decreases. On the other hand, as we broaden or shrink the scale of the criterion *duration*, the duration of the obtained plans increases. However, in this case we observe that when the scale broadens two times, the duration of the obtained plans decreases. This is an indication that a two-times broader scale is the critical scale for this criterion. We note finally that as the scales of the criteria *cost* and *duration* diversify from their critical scales, the planner reaches a solution significantly faster. This is because as these two criteria lose their strength, the effect of the criterion *length* on the planning process becomes more significant.

Table 9: Results for various criteria scales.

Experiment	m_{solved}	m_{time}	m_{length}	m_{cost}	$m_{duration}$
$A_{COST\ x\ 2}$	0.00%	-5.27%	-0.08%	3.10%	-0.93%
$A_{COST\ x\ 5}$	0.00%	-8.03%	1.95%	7.72%	-2.20%
$A_{COST\ x\ 0.5}$	0.00%	-3.38%	2.02%	7.24%	-1.14%
$A_{COST\ x\ 0.2}$	0.00%	-7.88%	1.85%	7.47%	-2.76%
$A_{DURATION\ x\ 2}$	0.00%	-4.39%	0.38%	3.02%	-0.27%
$A_{DURATION\ x\ 5}$	0.00%	-12.30%	0.01%	0.28%	5.24%
$A_{DURATION\ x\ 0.5}$	0.00%	-2.31%	2.78%	1.50%	11.67%
$A_{DURATION\ x\ 0.2}$	0.00%	-6.14%	1.55%	-1.82%	14.39%
$A_{LENGTH\ x\ 2}$	0.00%	228.51%	2.03%	-1.60%	1.18%
$A_{LENGTH\ x\ 5}$	-42.86%	2804.42%	3.89%	2.67%	-4.19%
$A_{LENGTH\ x\ 0.5}$	0.00%	-7.84%	-0.22%	-0.93%	0.44%
$A_{LENGTH\ x\ 0.2}$	0.00%	-7.88%	1.85%	7.47%	-2.76%

As for the criterion *length*, we observe that as its scale broadens, the planner needs more time to find a solution and, for greater broadenings, many problems become unsolvable in the specified limits of time and memory. Inversely, when we shrink this scale, problems are solved faster. This leads us to the conclusion that the critical scale for the criterion *length* is significantly narrower than the originally selected one.

Finally, we would like to lay emphasis on the fact that there is no ideal scale for a criterion. One could identify the critical scale for a criterion and for a specific problem, by repeatedly running the planner on this problem using different scales for the criterion and observing the quality of the resulting plans in terms of this criterion. However, this is not the best scale to be used. The reason is that the best scale for each criterion and for a specific problem is a very subjective matter that depends only on the evaluator's preferences, i.e. the person who is interested in the solution plan, and may be very different from the critical one. Extracting the preferences of the evaluator (criteria hierarchy, weights and scales) is a difficult problem that has been thoroughly studied in the area of decision-making and several methods have been proposed, e.g. interviews, questionnaires, etc.

GRT versus MO-GRT. We conclude our performance results by comparing GRT to MO-GRT. GRT can be considered a special case of MO-GRT, if all criteria have

zero weights, except for *length*, which has a weight equal to 1, and if the weight of the past plan is equal to 0, while the weight of the remaining plan is equal to 1.

Certainly, the conclusions drawn by the comparison depend on the weights and scales that will be used by MO-GRT. In order to retain a common reference in our experiments, we compare GRT to MO-GRT using again experiment A as a reference (Table 10).

Table 10: Comparison between the single-objective GRT and the MO-GRT in experiment A.

Experiment	m_{solved}	m_{time}	m_{length}	m_{cost}	$m_{duration}$
GRT	0.00%	-19.94%	0.16%	6.34%	-0.81%

As Table 10 shows, GRT is approximately 20% faster than MO-GRT. Note that this acceleration is greater than all accelerations encountered in all experiments. As far as the other criteria are concerned, GRT found plans of approximately equal length, higher cost and lower duration.

Summary and Future Challenges

This paper presents MO-GRT, a heuristic state-space STRIPS planner, which extends the single objective planner GRT with the ability to take multiple criteria into account. MO-GRT takes as input a user-defined hierarchy of criteria, which are considered important for the resulting plan, some preferences among them, in the form of weights, and a scale of allowable values for each basic criterion. As the experimental results have shown, different weights and scales result in plans of different quality, with respect to the criteria, and in different planning times. The work presented in this paper is the first attempt to apply multiple-criteria evaluation techniques in the area of domain-independent planning.

There are several challenges for future work in the area of multi-criteria planning. First of all, we intend to apply MO-GRT in other domains where multiple criteria are of interest and observe/analyze its performance and limitations. The creation of a reservoir of such domains is also in our goals.

An interesting extension of MO-GRT would be the development of a meta-system, which will analyze a planning problem in an attempt to identify the boundaries where the cost of solving the problem with respect to the various criteria lies. The development of such a system may require either some pre-processing planning (e.g. running the planner with marginal weights and collecting the solution plans) or domain-analysis techniques. This system would be useful to the evaluator, giving him the opportunity for a better understanding of the problem and helping him to set the scales for the various criteria.

Another challenge refers to the adoption of utility models (Haddawy, and Hanks, 1998). The extension of the techniques presented in this paper, so as to cover probabilities and utility models, is straightforward but is more costly from a computational point of view. Actually, a greater effort will be needed to construct the heuristic, since in that case, cost-vectors accompanied by their probabilities have to be computed for all the non-dominated ways to achieve a problem's facts and for all the possible worlds.

Finally, a last extension could be the special treatment of time. Currently, MO-GRT treats all criteria in a cumulative manner, however actions can be executed in parallel, so taking into account parallel action execution could result in more accurate plans.

References

Fikes, R.E.; Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189-208.

Fujimura, K. 1996. Path planning with multiple objectives. *IEEE Robotics and Automation Magazine* 3 (1):33-38.

Haddawy, P.; Hanks, S. 1998. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence* 14 (3): 392-429.

Keeney, R.L.; Raiffa, H. 1976. *Decisions with multiple objectives: Preferences and value tradeoffs*. John Viley & Sons.

Moraitis, P.; Tsoukias, A. 2000. Graph based representation of dynamic planning. In *Proceedings 14th European Conference on Artificial Intelligence*, 516-520. Ios Press.

Refanidis, I.; Vlahavas, I. 1999a. GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *Proceedings 5th European Conf. on Planning*, 346-358, Spinger.

Refanidis, I.; Vlahavas, I.; Tsoukalas, L. 1999b. On determining and completing incomplete states in STRIPS domains. In *Proceedings IEEE International Conference on Information, Intelligence and Systems*, 289-296, IEEE Press.

Refanidis, I.; Vlahavas, I. 2001. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research* 15:115-161.

Stewart, B.S.;White, C.C. 1991. Multiobjective A*. *Journal of the Association for Computing Machinery* 38 (4): 775-814.

Veloso, M. 1992. Learning by analogical reasoning in general problem solving. Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University.

Vincke, P. 1992. *Multi-criteria Decision Aid*, New York.

Williamson, M.; Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proceedings 2nd Int. Conf. on AI Planning Systems*, 176-181, AAAI Press.

Generating parallel plans satisfying multiple criteria in anytime fashion

Terry Zimmerman & Subbarao Kambhampati
Department of Computer Science & Engineering
Arizona State University, Tempe AZ 85287
Email: {zim, rao}@asu.edu

Abstract

We approach the problem of finding plans based on multiple optimization criteria from what would seem an unlikely direction: find one valid plan as quickly as possible, then stream essentially *all* plans that improve on the current best plan, searching over incrementally longer length plans. This approach would be computationally prohibitive for most planners, but we describe how, by using a concise trace of the search space, the PEGG planning system can quickly generate most, if not all, plans on a given length planning graph. By augmenting PEGG with a branch and bound approach the system is able to stream parallel plans that come arbitrarily close to a user-specified preference criteria based on multiple factors. We demonstrate in preliminary experiments on cost-augmented logistics domains that the system can indeed find very high quality plans based on multiple criteria over reasonable runtimes. We also discuss directions towards extending the system such that it is not restricted to Graphplan's scheme of exhaustively searching for the shortest step-length plans first.

I. Introduction

From a classical planning perspective a basic, multiple criteria optimization problem might entail finding a plan that optimizes two factors:

x: the number of time steps
y: the total 'cost' of the plan

Here the optimization itself will be with respect to some user-specified criteria involving x and y. Graphplan is a well-known classical planner that, in spite of the more recent dominance of heuristic state-search planners, is still one of the most effective ways to generate the so-called "optimal parallel plans". State-space planners are drowned by the exponential branching factors of the search space of parallel plans (the exponential branching is a result of the fact that the planner needs to consider each subset of non-interfering actions). However, there is no known practical approach for finding cost-optimal plans with Graphplan, let alone optimizing over some arbitrary weighting of time steps and cost. We describe and report on initial experiments with a Graphplan-based system that streams a sequence of plans that increasingly approach a user-specified optimization formula based on multiple criteria. This system, which we call Multi-PEGG, seeks to find the plan that comes closest to matching the user's preference expressed as a linear preference function on two variables. (e.g. $\alpha x + \beta y$, where x and y might be defined as above). As we'll discuss in Section V (future work) extending the system to handle more than two criteria is straightforward, as is implementation of criteria such as 'the least cost plan with no more than k steps'.

Consider first how a plan satisfying multiple criteria might be generated by Graphplan if computation time were not an issue. By alternating search episodes on the planning graph with extensions of the graph, Graphplan's algorithm is guaranteed to return the shortest plan in terms of time steps (where a step might include multiple actions that do not conflict). If Graphplan finds its shortest valid plan for the given problem on a k-level planning graph, a modest modification of the program could, in principal, find *all* possible valid k-length plans by conducting exhaustive search on the same planning graph.[1] The final set of plans could then be post-processed to find the best one in terms of any other optimization criteria giving us, for example, the least cost, k-length plan. However, not only is this approach computationally impractical for many problems/domains, but it can only handle a small subset of the multi-objective criteria one could envision. Such a system for example, could not satisfy a user request for the least-cost plan of any length.

In a naive attempt to extend the system capabilities so its scope includes plans of length greater than k, we might iteratively extend the planning graph, restarting the solution search for valid plans at each successive level. If we have a means of calculating 'cost' for the subgoal sets generated during the regression search, branch and bound techniques might be applied after finding the first valid plan to prune some of this search space. Nonetheless, this will clearly be an intractable approach for any problem of sufficient size to be of interest.

The PEGG (Pilot Explanation Guided Graphplan) planning system dramatically boosts Graphplan's ability to find step optimal plans by taking advantage of certain symmetries and redundancies in its search process [Zimmerman and Kambhampati, 2001, 2000]. We report here on preliminary work with extending PEGG in such a way that it leverages those planning graph related

[1] There are few subtleties involved in doing this. For example, care must be taken so that the subgoal sets generated in the regression search that directly leads to each valid plan are *not* memoized. The standard Graphplan goal assignment routine memoizes goal sets at each planning graph level as it backtracks.

symmetries to efficiently generate *all* plans of interest on any length graph. The 'Multi-PEGG' planner, which we focus on in this study, employs this capability together with a heuristic-based branch and bound strategy to generate a stream of increasingly higher quality plans (relative to the user's definition of quality). Given a variety of linear user preference formulas, we show that this approach can efficiently stream monotonically improving solutions for two different logistics domains augmented with action cost values.

The rest of this paper is organized as follows: Section II gives an overview of the PEGG system on which Multi-PEGG is based, and reports on its performance relative to Graphplan and one of the faster heuristic state space planners. Section III describes the extensions to PEGG that allow it to efficiently extract many, if not all, valid plans from a given length planning graph in reasonable time. Section IV then describes how Multi-PEGG exploits this capability along with branch and bound techniques to stream plans that come increasingly closer to a user-specified quality metric based on multiple criteria. Section V contains our conclusions and ideas for future work.

II. Using memory to expedite Graphplan's search for step-optimal plans

The approach we adopt to finding plans satisfying multiple criteria is rooted in the ability of the PEGG planner to efficiently find all valid plans implicit in a given length planning graph. The planning system makes efficient use of memory to transform the depth-first nature of Graphplan's search into an interactive state space view in which a variety of heuristics are used to traverse the search space [Zimmerman and Kambhampati, 2001, 2002]. It significantly improves the performance of Graphplan by employing available memory for two purposes: 1) to avoid some of the redundant search Graphplan conducts in consecutive iterations, 2) and (more importantly), to transform Graphplan's iterative deepening depth-first search into iterative expansion of a selected set of states that can be traversed in any desired order. We briefly review in this section the PEGG algorithm before describing how it can be adapted to find all plans on the graph.

The original motivation for the development of PEGG and the related planner that preceded it, EGBG [Zimmerman and Kambhampati, 1999], was the observation of redundancy in Graphplan's iterative-deepening solution search. Connections between Graphplan's search and IDA* search was first noted by Bonet and Geffner, 1999. One shortcoming of the standard IDA* approach to search is the fact that it regenerates so many of the same nodes in each of its iterations. It's long been recognized that IDA*s difficulties in some problem spaces can be traced to using too little memory. The only information carried over from one iteration to the next is the upper bound on the f-value. Given that consecutive iterations of search overlap significantly, we investigated methods for using additional memory to store a trace of the explored search tree in order to avoid repeated re-generation of search nodes. Once we have a representation of the search space that has already been explored, we can transform the way this space is extended in the next iteration. In particular, we can (a) expand the nodes of the current iteration in the order of their heuristic merit (rather than in a default depth first order) and/or (b) we can consider iteratively expanding a select set of states.

Although this type of strategy is too costly to implement in a normal IDA* search, the IDA*-search done by Graphplan is particularly well-suited to these types of changes as the kth level planning graph provides a compact way of representing the search space traversed by the corresponding IDA* search in its kth iteration. Realization of this strategy however does require that we provide an efficient way of extending the search trace represented by the planning graph, starting from any of the search states.

Consider the Figure 1 depiction of the search space for three consecutive Graphplan search episodes leading to a solution for a fictional problem in an unspecified domain. Represented here are just the substates that result from Graphplan's regression search on the ,X,Y,Z, goals, but *not* the mini CSP episodes that attempt to assign actions to each proposition in a state. Thus, each substate on a given planning graph level is linked to it's parent state and is composed of a subset of the parent's goals and the preconditions of the actions that were assigned. In each episode, we show substates generated for the *first* time in a unique shading and use the same shading when the states are regenerated *one planning graph level higher* in the subsequent search episode. A double line box signifies states that eventually end up being part of the plan that is extracted. As would be expected for IDA* search there is considerable similarity (i.e. redundancy) in the search space for successive search episodes as the plan graph is extended. In fact, the backward search conducted at level k + 1 of the graph is essentially a replay of the search conducted at the previous level k with certain well-defined extensions as defined in (Zimmerman and Kambhampati, 1999).

Certainly Graphplan's search could be made more efficient by using available memory to retain at least some portion of the search experience from episode n to reduce redundant search in episode n+1. This motivation was the focus of the EGBG system (Zimmerman and Kambhampati, 1999), which aggressively recorded the search experience in a given episode in a manner such that essentially *all* redundant effort could be avoided in the next episode. Although that approach was found to run up against memory constraints for larger problems, it suggests a potentially more powerful use for a much more pared-down search trace: leveraging the snapshot view of the entire search space of a Graphplan iteration to focus on the most promising areas. This transformation can free us from the depth-first nature of Graphplan's CSP search,

permitting us to move about the search space to visit it's most promising sections first -or even exclusively.

PEGG exploits the search trace it builds, extends, and prunes primarily for its view of the effective search space, and only secondarily to avoid some of the redundant search

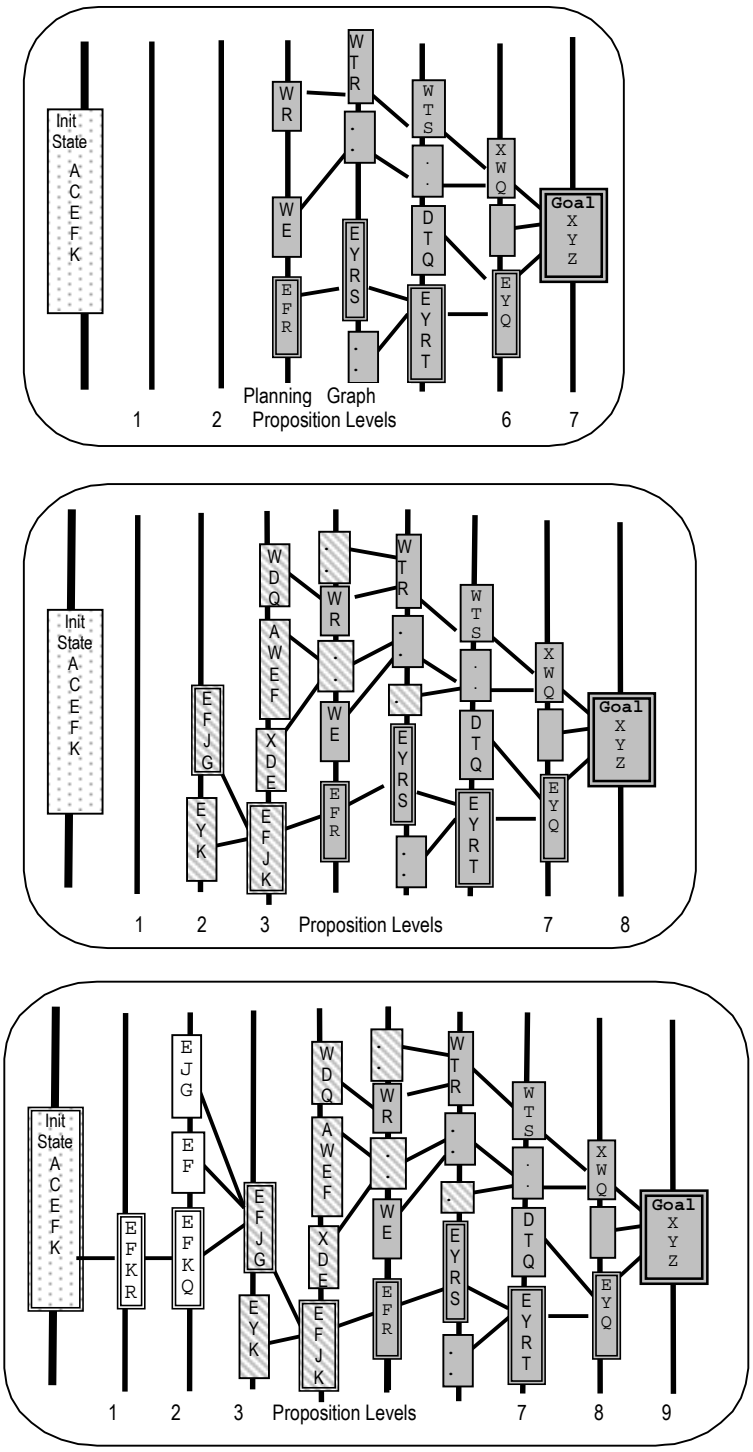


Figure 1. Graphplan's search space: 3 consecutive search episodes on the planning graph

across episodes. The PEGG algorithm for building and using a search trace retains Graphplan's iterative nature but significantly transforms its search process. We make the following two informal definitions before describing the algorithm developed to transform Graphplan's search:

Search segment: a node-state as generated during Graphplan's regression search from the goal state (which is itself the first search segment), indexed to a specific level of the planning graph. Key content of a search segment S_n at plan graph level k is the proposition list for the state, a pointer to the parent search segment (S_p), and the actions assigned in satisfying the parent segments goals. The last information is needed once a plan is found in order to extract the actions comprising the plan from the search trace.

Search trace (ST): the entire linked set of search segments (states) representing the search space visited in a Graphplan backward search episode. It's convenient to visualize it as a tiered structure with separate caches for segments associated with search on plan graph level k, k+1, k+2, etc. We also adopt the convention of numbering the ST levels in the reverse order of the plan graph; the top ST level is 0 (it contains a single search segment whose goals are the problem goals) and the level number is incremented as we move towards the initial state. When a solution is found the search trace will necessarily extend from the highest plan graph level to the initial state, and the plan actions can be extracted from the linked search segments in the ST without unwinding the search calls as Graphplan does.

We also define some processes:

Search trace translation: For a search segment in the ST associated with plan graph level j after search episode n, associate it with plan graph level j+1 for episode n+1. Iterate over all segments in the ST. The fact that search segments are mapped onto the plan graph helps minimize the memory requirements. In order to pickup Graphplan's search from any state in the trace, the number of valid actions for the state goals and their mutex status must be known. The simple expedient of successively linking the search segment to higher plan graph levels in later search episodes makes this bookkeeping feasible.

Visiting a search segment: For segment S_p at plan graph level j+1, visitation is a 3 –step process:

1. Perform a memo check to ensure the subgoals of S_p are not a nogood at level j+1
2. Initiate Graphplan's CSP-style search to satisfy the segment subgoals beginning at level j+1. A child search segment is created and linked to S_p (extending the ST) whenever S_p's goals are successfully assigned.
3. Memoize Sp's goals at level j+1 if all attempts to consistently assign them fail.

We claim, without proof here, that as long as *all* the

segments in the ST are visited in this manner the planner is guaranteed to find a 'step-optimal' plan in the same search episode as Graphplan (though the number of actions in the plan may differ).

The entire PEGG trace building and search process is detailed [Zimmerman and Kambhampati, 2001, 2002] and we only outline it here. The search process is essentially 2-phased: a promising state from the ST must be selected, then depth-first CSP-type search on the state's subgoals is conducted. If the CSP search fails to find a plan, the planner selects another ST search segment to visit. Our work with a variety of different search trace architectures has highlighted the importance of keeping the search trace small and concise, both due to memory constraints and because the search effort expended in non-solution bearing episodes increases in direct proportion to the number of segments in the ST. We've employ a variety of CSP speedup techniques for the Graphplan style portion of the search process and find that the benefits are compounded because they greatly reduce the number of states visited - and hence tracked in the ST. Chief amongst these methods are explanation based learning (EBL), dependency directed backtracking, domain preprocessing and invariant analysis, and a bi-level plan graph.

As described in [Zimmerman and Kambhampati, 2001, 2002], the search trace provides us with a concise state space view of PEGG's search space, and this allows us to exploit the 'distance based' heuristics employed by state space planners such as HSP-R (Bonet and Geffner, 1999) and AltAlt (Nguyen and Kambhampati, 2000). Two of the approaches for employing these heuristics in PEGG that we have investigated are:

- Ordering the ST search segments according to a given state space heuristic and visiting all of them in order (we term this PEGG-b [2])
- Ordering the ST search segments according to a given state space heuristic and retaining only the 'best' fraction for visitation (PEGG-c)

The first approach maintains Graphplan's *guarantee* of step optimality but focuses significant speedup only in the final search episode. The second approach sacrifices the guarantee of optimality in favor of pruning search in *all* search episodes and bounds the size of the search trace that is maintained in memory. As we've reported previously, optimal length plans are generally found, regardless. For this study, Multi-PEGG is run only under the PEGG-b conditions (entire search space visited subject to branch & bound constraints) and we defer further discussion of the PEGG-c to the future work assessment of Section V.

Table 1 compares the performance of PEGG operation to standard Graphplan as well as Graphplan enhanced with the CSP speedup techniques that have been incorporated in PEGG (EBL, DDB, domain preprocessing, etc.). Clearly the enhancements alone have a major impact on standard Graphplan's performance, significantly extending the range of problems it can solve. Focusing on the PEGG-b column its ability to leverage its inter-episodic memory becomes apparent. PEGG-b accelerates planning, by factors of up to 300 over standard Graphplan and 2 - 14x over even the enhanced Graphplan.

When running in this mode, PEGG uses the 'adjusted-sum' distance heuristic described in [Nguyen and Kambhampati, 2000] to move about the search space represented in the ST. Summarizing their description: The heuristic cost *h(p)* of a single proposition is computed iteratively to fixed point as follows. Each proposition *p* is assigned cost 0 if it's in the initial state and ∞ otherwise. For each action, *a*, that adds *p*, *h(p)* is updated as:

h(p) := min{h(p), 1+h(Prec(a) }

where *h(Prec(a))* is computed as the sum of the h values for the preconditions of action a.

Define *lev(p)* as the first level at which *p* appears in the plan graph and *lev(S)* as the first level in the plan graph in which all propositions in state *S* appear and are non-mutexed with one another. The adjusted-sum heuristic may now be stated:

$$h_{adjsum}(S) := \sum_{p_i \in S} \cos t(p_i) + lev(S) - \max_{p_i \in S} lev(p_i)$$

It is essentially a 2-part heuristic; a summation, which is an estimate of the cost of achieving *S* under the assumption that its goals are independent, and an estimate of the cost incurred by negative interactions amongst the actions that must be assigned to achieve the goals. (Due to space considerations, we limit our experimentation here to only this distance heuristic.)

As discussed in [Zimmerman and Kambhampati, 2001, 2002], PEGG-b exhibits speedup over Graphplan in spite of the fact that it revisits (but *doesn't* regenerate) every state that Graphplan generates in each non-solution bearing search episode. One primary sources of its advantage lies in the fact that any state in the ST from the previous episode can be extended in the new episode without incurring the search cost needed to regenerate it. If a state in the deepest levels can be extended to the initial state, we will have found a solution while completely avoiding all the higher level search required to reach it from the top level problem goals. Hereafter we refer to a search trace segment that is visited in the solution episode and extended via backward search to find a valid plan as a *seed segment*. Thus, to the extent that the search heuristic identifies a seed segment deep in the ST in the solution episode, PEGG will greatly shortcut the search in what is often the most costly of Graphplan's iterations.

In the next section, we describe an extension to PEGG that enables the system to find (in most cases) *all* step-optimal plans implicit in a given planning graph. This will prove to be key capability in order for Multi-PEGG to generate plans satisfying multiple optimization criteria.

[2] The name scheme for PEGG operating in various modes used in [Zimmerman and Kambhampati, 2001, 2002] is retained here to avoid possible confusion.

Problem	Stnd GP cpu sec	GP-e (enhanced Graphplan) cpu sec (steps/acts)	PEGG-b heuristic: adjsum cpu sec (steps/acts)	PEGG-c heuristic: adjsum cpu sec (steps/acts)	Alt Alt (Lisp version) cpu sec (/ acts) heuristics: adjusum2 combo
bw-large-B	234.0	101.0 (18/18)	12.2 (18/18)	9.4 (18/18)	87.1 (/ 18) 20.5 (/28)
bw-large-C	~	~	~	60.5 (28/28)	738 (/ 28) 114.9 (/38)
bw-large-D	~	~	~	460.9 (36/36)	2350 (/ 36) ~
Rocket-ext-a	846	39.8 (7/36)	2.8 (7/34)	1.1 (7/34)	43.6 (/ 40) 1.26 (/ 34)
Rocket-ext-b	~	27.6 (7/36)	2.7 (7/34)	2.7 (7/34)	555 (/ 36) 1.65 (/34)
att-log-a	~	31.8 (11/79)	2.6 (11/56)	2.2 (11/62)	36.7 (/56) 2.27(/ 64)
att-log-c	~	~	~	22.9 (12 /57)	53.3 (/ 47) 19.0 (/67)
Gripper-8	~	28.8 (15/23)	16.6 (15/23)	8.0 (15/23)	6.6 (/ 23) *
Gripper-15	~	~	47.5 (36/45)	16.7 (36/45)	14.1 (/ 45) 16.98 (/45)
Gripper-20	~	~	~	44.8 (40/59)	38.2 (/ 59) 20.92 (/59)
Tower-7	~	114.8 (127/127)	14.3 (127/127)	1.1 (127/127)	7.0 (/127) *
Tower-9	~	~	118 (511/511)	23.6 (511/511)	121(/511) *
Mprime-1	17.5	4.8 (4/6)	3.6 (4/6)	2.1 (4/6)	722.6 (/ 4) 79.6 (/ 4)
Mprime-16	~	54.0 (8/13)	35.2 (8/13)	5.9 (4/6)	~ ~
8puzzle-1	2444	95.2 (31/31)	39.1 (31/31)	9.2 (31/31)	143.7 (/ 31) 119.5 (/39)
8puzzle-2	1546	87.5 (30/30)	31.3 (30/30)	7.0 (30/30)	348.3 (/ 30) 50.5 (/ 48)
8puzzle-3	50.6	19.7 (20/20)	2.7 (20/20)	1.8 (20/20)	62.6 (/ 20) 63.3 (/ 20)
aips-grid1	312	66.0 (14/14)	34.9 (14/14)	8.4 (14/14)	739.4 (/14) 640.5 (/14)
aips-grid2	~	~	~	129.1 (26/26)	~ ~

Table 1 PEGG performance vs. Graphplan, enhanced Graphplan and a BSS heuristic planner

GP-e: Graphplan enhanced with bi-level PG, domain preprocessing, EBL/DDB, goal & action ordering
PEGG-b: Same as PEGG, all segments visited as ordered by adjsum heuristic
PEGG-c: bounded PE search, only best 20% of search segments visited, as ordered by adjsum heuristic
Parentheses next to cpu time give # of steps/ # of actions in solution
All planners in Allegro Lisp, runtimes (excl. gc time) on Pentium 500 mhz, Linux, 256 M RAM
"adjusum2" and "combo" are the most effective heuristics used by AltAlt
~ indicates no solution was found in 30 minutes * indicates problem wasn't run

III Extracting *all* valid plans with PEGG

As discussed in the introduction, extracting all valid plans from even the k-level planning graph, where k is the first level at which a problem solution can be found, is in general intractable for Graphplan. Indeed, no existing planner efficiently does this. We describe here a version of PEGG, which we call PEGG-ap (All Plans) that can in fact efficiently generate all such plans in reasonable time for problems that are not highly solution dense and can stream an arbitrarily large number of them even when there are thousands. It's the combination of PEGG's search trace and the planning graph that make this a feasible proposal for PEGG.

Consider the depiction of Graphplan's search space in the solution episode (third graph) of Figure 1. This corresponds to the ST as it exists immediately after the first plan is found. At this point we've provably shown that each state (set of subgoals) corresponding to the sets of assigned actions in a step of this plan can be extended to the initial state via Graphplan's CSP-style search. These states are the nine speckled search segments in the figure and we will hereafter refer to any such state as a *plan state*. In effect then, such a state at level m can be seen as the root node of a subtree with *at least one* branch that extends from level m to the initial state. We will call such a subtree a *plan stem* (or just stem) and observe that there may be many valid plans implicit in the given planning graph that have the same plan stem as their base. Now consider a planning system which seeks to find all valid plans on a planning graph and that can keep track of such plan stems each time it finds a new plan. If the system can efficiently check during the regression search to see if the set of subgoals, *S*, to be satisfied at a given level m corresponds to one of these states, it has a powerful means of shortcutting that search. Whenever *S* corresponds to one of the plan stem nodes in memory the planner will have found a new plan with a head consisting of the actions/steps assigned in regression search to level m and a tail consisting of the actions/steps corresponding to the plan stem in memory. It can then immediately backtrack in search of other plans.

The fact that PEGG conducts its search on a planning graph suggests an efficient approach for retaining in memory the states associated with a valid plan: the same caches used to memoize states that *cannot* be consistently satisfied during regression search (i.e. 'nogoods') can be used to memoize the states in the extracted plan. Like Graphplan, PEGG's memo-checking routine checks these planning graph level-specific caches anyway before attempting to assign a set of subgoals in CSP fashion. PEGG-ap, has a modified memo saving routine so that when a valid plan is found, it memoizes each plan state at its associated planning graph level, and includes a pointer to the search segment in the ST. The PEGG-ap memo-checking routine differentiates between a nogood memo and a plan state memo such that when a search state matches a plan memo (from some plan already identified), the routine returns a pointer to the relevant search segment in the ST. This enables PEGG-ap to construct a new plan(s) without further search.[3] Note that since all search segments that are part of a valid plan are anyway contained in the ST, it is not necessary to actually store each plan so generated. As long as we maintain a list of the last search segment in a plan tail (i.e. the state whose subgoals are subsumed by the initial state) the upward-linked structure of the ST allows us to extract all identified plans from it on demand.

PROBLEM	TOTAL PLANS	RUN TIME 1ST PLAN	RUN TIME ALL PLANS	SIZE OF ST (no. of states) After 1st plan / After all plans
BW-LARGE-A	1	1.3	2.9	52 / 107
HUGE-FCT	84	9.3	26.6	6642 / 16,728
FERRY6	384	15.8	17.2	377 / 427
GRIPPER8	1680	17.0	32.5	7670 / 10,730
TOWER6	1	1.9	2.3	315 / 440
EIGHT1	12	40.1	75.0	18,650 / 29,909
ROCKET-EXT-A	(>2073)	2.9	(> 14,000)	188 / (238)
ROCKET-EXT-B	1111	1.1	77.0	194 / 2200
ATT-LOG-A	1639	2.9	2407	279 / 818

Table 2 PEGG-ap experiments with extracting *all* plans at the first solution level of the planning graph
Values in parentheses are partial results reported at the time the run was terminated. All planners in Allegro Lisp, runtimes (excl. gc time) in cpu seconds on Pentium III, 900 mhz, Windows 98, 128 M RAM

The performance of PEGG-ap on a sampling of benchmark planning problems is reported in Table 2. The system was set to search in the PEGG-b mode; all ST search segments are ordered and visited according to the 'adjusted-sum' heuristic. The first column of values reports the total number of step-optimal plans generated at the planning graph level at which the first problem solution was found. Clearly, the solution density varies greatly across domains and problems, from the Tower of Hanoi domain that can only have one solution to the logistics domains that may have thousands of valid optimal plans implicit in the planning graph at the solution level. Columns 3 and 4 report run times in cpu seconds to find the first plan and all plans respectively, and the figures testify to the effectiveness of this approach in extracting the remaining plans once the first plan has been found. For example, on the HUGE-FCT problem it takes PEGG-ap 9.3 seconds to generated the first solution and then just over 17 seconds to find the remaining 83 on the 18-level planning graph. The first solution to GRIPPER8 is found in 17 seconds and then the remaining 1679 solutions are generated within another 16 seconds. Many logistics domains problems are so solution dense however that there are thousands of step-optimal plans on the planning graph at the solution level. In the case of ROCKET-EXT-A, for example PEGG-ap had streamed over 2000 plans in 3 ½ hours when the run was terminated.

The fifth column provides a measure of the additional memory required in order for PEGG-ap to extract all step-optimal plans as compared to just the first plan found. We compare here the size of the search trace at the time the first plan is generated with its size after all plans have been found. As expected, the ST grows as more of the states are visited in an attempt to find other plans, but the growth is not linear in the number of plans. This is a reflection of the fact that for most domains/problems plans often share many of the same ST states. The number of search segments (states) in the ST increases by a factor of 11 in the worst case here, but on average the increase is a factor of 2 larger. In no case has this memory demand exceeded the available swap space on the machine used.

IV Streaming plans based on multiple optimization criteria

Up to this point, all versions of PEGG we've discussed are capable of optimizing the number of plan steps. This ability is inherited from the IDA* nature of Graphplan's search process (The connections between Graphplan's search and IDA* was first noted by Bonet and Geffner, 1999.) In order for Multi-PEGG to also handle other optimization criteria, we must have a means of estimating the 'cost' of a achieving a state in terms of the criteria. We start by assigning propositions in the initial states a cost of *zero* and an execution cost for each action. Since PEGG conducts regression search from the problem goals, the cost of reaching those goals from any state generated during the search (e.g. the states in the ST) is easily tracked as the cumulative cost of the assigned actions up to that point. Estimating the cost of reaching a given state from the initial state however, is problematic. To evaluate that cost we need to propagate the costs from the initial state to the state using the mutual dependency between propositions and actions. Specifically, the cost to achieve a proposition

[3] A search segment can be a stem root for more than one valid plan since there may be more than one consistent assignment of actions satisfying its goals.

depends on the cost to execute the actions supporting it, which in turn depends on the costs to achieve propositions that are their preconditions. The planning graph is well suited to represent the relation between propositions and actions, and we will make heavy use of it.

There are two measures of action and state cost that we calculate and propagate in Multi-PEGG:

- *Max cost*: the value of the proposition with the maximum cost in a set of propositions (a state or the preconditions of an action).
- *Sum cost*: the sum of the costs of all propositions in a set

The first measure is most accurate when all preconditions of an action (state) depend on each other and the cost to achieve all of them is equal to the cost to achieve the costliest one. This measure never overestimates the cost and is *admissible*. The second measure is most accurate when a state or all preconditions of an action are independent. Although clearly inadmissible, it has been shown in [11; 2] to be more effective than the *max* measure. Note that the sum cost will always decrease for an action when the cost of one of its preconditions improves, but this is not guaranteed for max cost. As described below we will make use of these measures both separately and in combination in deciding which states to expand during search.

In seeking a 'compromise' estimate of the true cost of reaching the initial state from a given state, we have considered linear combinations of the max and sum measures. Noting that the last two terms of the adjusted-sum heuristic (see section II) provide a measure of the inter-dependence of the propositions in a state, we experimented with using it as a weighting. The following cost estimate for a state *S*, which we will call *adjusted-combo,* has proven effective for the Multi-PEGG search process we will describe below:

$$cst_{adj-combo} = \left[\frac{lev(S) - \max_{p_i \in S} lev(p_i)}{glev(S) - 1}\right] sum(S) + \left[1 - \frac{lev(S) - \max_{p_i \in S} lev(p_i)}{glev(S) - 1}\right] \max(S)$$

where: *lev(S)* and *lev(p)* are as defined in section II, and *glev(S)* is the planning graph level at which state S is currently being evaluated.

Note that since no state *S* will ever be generated in regression search at a planning graph level lower than *lev(S),* the two weighting terms (in brackets) will always sum to one. This cost estimate has the desired property that the higher the degree of negative interactions between the subgoals in *S,* the larger the fraction of the estimate comes from summing the cost of its subgoals. This is clearly an inadmissible heuristic since it can overestimate the cost of a state, but this is of somewhat less concern since Multi-PEGG seeks to stream plans of increasing quality.

We also must confront the issue of normalizing the cost component to the length component when they are combined in a user's linear preference formula. The intent of a preference formula such as $\alpha\ length + \beta\ cost$ will not be met if there is no base upon which they can be compared. Ideally, we'd like to normalize each component over its optimal value, but in general, we don't know those values. However, as described below, Multi-PEGG in fact first finds a step-optimal plan and then seeks to find a better plan with respect to the user's preference. As such, at the point where it needs a value for plan quality in order to conduct branch and bound search, it has the *optimal* plan length and *one possible* plan cost in hand. When generating the quality value, *q* for a candidate plan we use these base values (*opt-length* and *base-cost* , respectively) to perform a rough normalization of the actual plan parameters (*length* and *cost*) in Multi-PEGG as follows:

$$q = \alpha \frac{\text{length}}{\text{opt–length}} + \beta \frac{\text{cost}}{\text{base–cost}}$$

We can now give an overview of the high-level algorithm used by Multi-PEGG to stream plans that increasingly approach *q,* a specified optimization formula involving more than just plan length:

1. Find the first valid plan -which will be step optimal- using PEGG's approach for conducting search using a search trace. Memoize its constituent states as successful plan states and return the plan to the user. Whenever the planning graph is extended, propagate not only mutex information but also action and proposition *max* and *sum* cost information.
2. With a valid plan in hand, determine it's quality value based on the user-specified criteria, *q*.
3. Define the search space for the next search episode in the following manner: Sort the remaining search segments (states) in the ST based on their *q* criteria. Plan length is set by the current length of the planning graph (say, *k*) and estimates of a state's cost are made based on the propagated cost of its subgoals using the *adjusted-combo* formula.
4. Seek increasingly 'higher quality' plans by conducting branch and bound search (using the *q* value of the best plan found) on the sorted ST states. Any candidate state is *visited* (as defined in section II) as long as its estimated *q* value is less than that of the current best plan. New plans are generated in the manner described for PEGG-ap; either by reaching the initial state or an existing plan state. Whenever the branch and bound finds a lower cost plan, return it to user, memoize its plan states, and update the bounding *q* value.
5. When the branch & bound search space is exhausted at level k, extend the planning graph (propagating cost information), translate the ST up one level, and sort the search states as described in step 3 -with two additions:
   a. Filter from the search space for this episode any state that does not have a decreased *sum* cost

value. (If the cost has not decreased there is no way that it can be extended to a lower cost plan than the current best.)

b. Each state *S,* visited in the previous episode at associated planning graph level *k* that does not extend to a plan effectively provides an updated estimate of *lev(S).* Instead of the original *lev(S)* value, which is the first level at which the propositions are binary non-mutex, we now have an n-ary non-mutex level estimate, which is just *k.*

6. Return to step 4.

This algorithm could of course go on seeking a better plan indefinitely, so in practice we enforce a maximum runtime.

To date Multi-PEGG has been tested on three classical problem domains that we modified to enable testing of its ability to handle multi-criteria.

ROCKET domain

The standard version of this highly parallel logistics domain involves multiple rockets that fly between locales carrying cargo and people. We added cost values to the domain actions as follows:

- rockets' MOVE action> 4
- REFUEL> 3
- LOAD and UNLOAD actions> 1

The benchmark ROCKET-EXT-A and B problems involve 2 rockets, 4 locales, and 10 people and cargo items that must reached goal locations. For both problems Graphplan finds a step-optimal plan of length 7, (which involves using both rockets) but there are a *large* number of such step optimal plans on the 7-level planning graph (see Table 2) and the number of actions in them may vary between 30 and 36. For this fairly simple problem structure it's straightforward to manually determine the optimal plans in terms of actions or cost; if only one rocket is used the goals can be reached in two fewer rocket trips, but it requires one additional plan step. Beyond 8 steps no other cost reductions are achievable.

Table 3 reports on Multi-PEGG's performance in seeking an optimal plan based on different linear combinations of the plan length and plan cost criteria. Here we attempt to give a feel for the dynamic nature of the plan streaming by reporting for each user preference formula , the plan length and cost and its calculated *q* value for the first plan found, and then after 30, 120, and 1200 cpu seconds of runtime. (We don't report values for the *1.0 L + 0 C* formula since this is basic Graphplan's bias. Because cost is absent from the optimization expression, all plans found at the first solution level will have equal 'quality'.) The table reveals several interesting characteristics of Multi-PEGG's search process. Once the first plan is found on the 7-level planning graph, the branch and bound search for a lower cost plan on that graph is quite effective in pruning the search space. Whereas PEGG-ap was still searching for all possible plans at level 7 after 14,000 seconds, Multi-PEGG, after 1200 seconds, completes its search at level 7, extends the planning graph, and conducts search on the 8 level graph for all but the first row optimization criteria. The higher the cost weighting of the criteria, the more the search is pruned on a given planning graph level. The inadmissible nature of the *adjusted-combo* cost heuristic is manifest in the fact that the .8L + .2C and .5L + .5C formulas find some slightly lower cost plans on the 7-level graph than the two formulas with higher cost weightings. However the user's preference appears to be reasonably served for these latter two formulas in that they move on fairly quickly to find some much higher quality (lower *q* value) plans -at least based on *their* criteria- on the 8-level planning graph.

The ROCKET domain problem provides limited exercise for the type of multiple criteria optimizing that Multi-PEGG does, so we look next at a more complex logistics domain involving more than one mode of transportation with different associated costs.

ATT LOGISTICS domain

The standard version of this domain involves two modes of transporting packages; via airplane and via truck. However, the trucks can only operate within a city (hauling packages from the Post Office to the airport) and the airplanes are used to fly between cities. We've extended this domain by not only giving costs to the actions, but enabling trucks to travel between cities that are within range of their fuel capacity. They must refuel at each such city. (For simplicity, we've not introduced actual refueling actions for airplanes, but it's straightforward to do so). The trucks are constrained from traveling directly to any city by

Optimization Criteria L: length C: cost	**1st Plan** [step length/cost]	*q* val	cpu sec.	**Best plan at 30 sec** [step length/cost]	*q* val	**Best plan at 2 min.** [step length/cost]	*q* val	**Best plan at 20 min.** [step length/cost]	*q* val
.8 L + .2 C	[7 / 56]	1.0	3	[7 / 52]	.98	[7 / 52]	.98	[7 / 50]	.97
.5 L + .5 C	[7 / 56]	1.0	3	[7 / 52]	.96	[7 / 52]	.96	[8 / 49]	.95
.2 L + .8 C	[7 / 56]	1.0	3	[7 / 56]	1.0	[7 / 56]	1.0	[8 / 45]	.89
0 L + 1.0 C	[7 / 56]	1.0	3	[7 / 56]	1.0	[8 / 49]	.86	[8 / 45]	.80

Table 3. Multi-PEGG streaming of plans on the ROCKET-EXT-A problem, modified to include action costs.
All planners in Allegro Lisp, runtimes (excl. gc time) in cpu seconds on Pentium III, 900 mhz, Windows 98, 128 M RAM

a 'NEXT-TO' fact added to the 'DRIVE' operator and a set of facts in the initial condition that prescribe which cities are directly next to each other. The cost values for actions are as follows:

- LOAD-TRUCK, UNLOAD-TRUCK> 1
- LOAD-AIRPLANE, UNLOAD-AIRPLANE> 1
- DRIVE-TRUCK1 (local, in-city trip)> 1
- DRIVE-TRUCK2 (inter-city trip)> 3
- REFUEL-TRUCK (needed inter-city only) 1
- FLY> 20

This cost structure is such that, depending on such things as where the truck and package(s) are located in a city, whether their destination is the airport or a post office of a distant city, and how many times a truck must be refueled, transporting the cargo via truck may be cheaper than flying. Note that delivery via truck could also take fewer steps than via airplane because transfer of the cargo from truck to airplane is avoided.

The original benchmark ATT-LOG-A problem that we focus on here features 8 packages to be transported, 3 cities (LA, PGH, BOS) each having one airport and one post office, 1 truck in each city (initially), and both airplanes are in one city. The step-optimal plan for the standard problem is 11 steps and PEGG-ap finds that there are plans ranging from 52 to 76 actions on this 11-level planning graph. (In terms of our introduced cost structure the least cost, 11-step plan would have a value of 128).

Our modified ATT-LOG-A problem retains *all* the original parameters except that we introduce connected cities linking the three destination cities (and thus permitting truck travel) as follows:

- 4 cities between BOS and PGH
- 6 cities between PGH and LA
- 6 cities between BOS and LA

Each of these connecting cities contains an airport (but no post office) so airplanes can also visit them and, feasibly load/unload cargo. We designed the routing structure so that, in combination with the cost structure, truck transportation of cargo will only provided a cost advantage between the cities of BOS and PGH, albeit at the expense of time steps. We note that the additional transportation routes increases the branching factor of this problem considerably, so that although PEGG-ap extracts all step-optimal plans of the original problem within about 40 minutes, it is unable to do so in twice that long on our modified version.

Table 4 reports the performance of Multi-PEGG on this problem for the same optimization formulas and runtime intervals discussed for Table 3. Here there is much greater variation in the quality of the streamed plans due to the more complex structure of the logistical domain. Broadly speaking, the streaming process on this problem has two main phases once the first, step-optimal plan is found; 1) optimizing over the cost of various action sets in alternative 11-step plans 2) searching beyond 11 steps for longer, but less costly plans that use inter-city truck transportation between PGH and BOS instead of airplanes. The branch and bound on plan cost again greatly helps in pruning the search space, as Multi-PEGG begins examining plans of greater than the step-optimal length within 20 minutes for three of the four optimization formulas. For the formulas in the last two rows of the table, Multi-PEGG in fact examines 13-step plans and greatly improves on its least cost 11-step plan by finding some that use the PGH truck to transport three packages to BOS instead of flying them.

The reported results also indicate that, while increasing the bias towards low cost plans causes a more rapid move in this direction for the first two formulas, the trend does not continue with the third formula (compare plan cost trends for these formulas in columns 3 or 4). This is probably due to the complex interactions between how the ST search space is visited (which is directed by the cost heuristic) and the subsequent memoization of both failing nodes and successful plan stem nodes.

V Conclusions and Future Work

We have conducted an investigation into the feasibility of

Optimization Criteria L: length C: cost	**1st Plan** [step length/cost] *q* val	cpu sec.	**Best plan at 30 sec** [step length/cost] *q* val	**Best plan at 2 min.** [step length/cost] *q* val	**Best plan at 20 min.** [step length/cost] *q* val
.8 L + .2 C	[11 / 208] 1.0	12	[11 / 182] .98	[11 / 166] .97	[11 / 128] .94
.5 L + .5 C	[11/ 208] 1.0	12	[11 / 166] .95	[11 / 144] .94	[11 / 128] .90
.2 L + .8 C	[11 / 208] 1.0	12	[11 / 180] .96	[11 / 160] .95	[13 / 111] .78
0 L + 1.0 C	[11 / 208] 1.0	12	[11 / 166] .91	[13 / 115] .80	[13 / 107] .71

Table 4. Multi-PEGG streaming of plans on the ATT-LOG-A problem, modified to include action costs.

All planners in Allegro Lisp, runtimes (excl. gc time) in cpu seconds on Pentium III, 900 mhz, Windows 98, 128 M RAM

streaming parallel plans satisfying multiple criteria using a Graphplan-based planning system. Our preliminary work shows that Multi-PEGG's use of a concise search trace can be exploited to allow it to efficiently generate a stream of plans that monotonically approach a user's preference for plan quality when expressed as a linear preference function on two variables. On the admittedly limited number of problems examined to date, Multi-PEGG is not only capable of finding the least cost step-optimal plan, but it finds longer length plans that come closer to satisfying the multi-objective optimization criteria.

Extending the current system to handle different optimization criteria and more than two does appears to be a straightforward task. Each such criterion requires a suitable estimation function, and the 'cost' values must be propagated in the planning graph separately. However, the approach to ordering states in the ST according to a multi-variable linear preference functions remains unchanged. It is also not a difficult undertaking to extend the type of criteria the user can employ to such things as 'I am not interested in plans costing over x' or 'Give me only plans shorter than length y'.

Overcoming the make-span bias of Multi-PEGG

In spite of the early success of the approach reported in this paper, it clearly has some disadvantages. It inherently starts with a step-optimal plan and, with some help from branch and bound techniques, searches on incrementally longer planning graphs streaming it's current best plan as it does so. If the user's primary plan quality criteria is cost, not length, and the types of low cost plans that are likely to be of interest are many steps longer than the shortest length plan, this approach could be unsatisfactory. Although we recognized this limitation early in the investigation, we also had in mind two major augmentations that might well overcome it, and so proceeded with a test of the simpler system reported here. We discuss these two augmentations to Multi-PEGG next.

Liberation from Graphplan's level-by-level search

There is in fact nothing formidable that *requires* Multi-PEGG to finish its search on a given planning graph level before considering possible plans on extensions of the planning graph. The search trace again proves to be very useful in this regard. Once the first valid plan has been found and a plan quality value established for subsequent branch and bound search, the ST can be translated up any desired number of levels (subject to the ability to extend the graph correspondingly and propagate the cost values) and used in a search for plans of arbitrary length. Referring back to Figure 1, this is equivalent to translating the ST tree of the third search episode pictured upward on the planning graph so that the XYZ root node now lies on some level higher than 9. If we then assess the multi-criteria *q* values for the search segments (states) in the ST at these higher levels we can co-mingle them with the same search segments from lower levels and *order all of them together* for visitation according to our plan quality formula. To the extent that we have an effective estimation formula for identifying the lowest cost plans, this will essentially enable Multi-PEGG to concurrently consider multiple length plans in its branch and bound search for a better plan.

This would be a prohibitive idea in terms of memory requirements if we had to store multiple versions of the ST, but we can retain only the one version of it and simply store any level-specific cost and heuristic information in its search segments as values indexed to their associated planning graph levels. Interesting problems that arise include such things as what range of plan lengths should be considered at one time and how to avoid having to deal with plans with steps consisting entirely of 'persists' operators.

Shortcutting the search in a given episode

Of the two modes for employing distance heuristics discussed in section II, we have only reported the performance of Multi-PEGG when it visits *all* states in the ST (i.e. PEGG-b mode), modulo the branch and bound process. It's also possible to augment the branch and bound pruning of search by screening from consideration those states that do not meet some threshold criteria based on a distance heuristic. Such states generated in Graphplan's regression search hold little or no promise of being extended into a solution, yet their inclusion in the search trace means PEGG will have to expand them eventually in each intermediate search episode. We have found that the distance-based heuristics are effective in identifying such states, and have experimented with various threshold options for restricting those maintained in the ST. Although such filtering of the search space forfeits the *guarantee* that PEGG will return a step-optimal solution, in practice we find that that even restricting the active ST to the heuristically best 10-15% of the generated states has no impact on the quality of returned plans. When PEGG operates in this mode, (tagged as 'PEGG-c' in Table 1) there is a dramatic reduction of both the size of the working ST and the time spent in non-solution bearing search episodes. As indicated, PEGG-c solves many more problems than either standard or enhanced Graphplan (GP-e) and exhibits speedups of 40x or more over GP-e where both find solutions. The table also reports the length of the plans produced (in terms of steps and actions). In all cases, PEGG-c finds a plan of equivalent step-length to the Graphplan optimal plan.

The intuition for Multi-PEGG is that, besides looking for the 'next best plan' we only want to visit a search segment in the ST that has a high likelihood of being extended into a valid plan. Of course, this may also screen out some of the longer length but lower cost plans that we may be interested in, so this is an empirical issue that needs to be investigated.

Bounding the length of plans that need to be considered

Another interesting issue associated with this approach to optimizing over multiple criteria is whether we can assess *when* the streaming process can be terminated due to no (or low) possibility of improving on the current best plan. The planning graph may again prove to be a useful structure for deducing such bounds on the search process.

References

Zimmerman, T. and Kambhampati, S. 1999. Exploiting Symmetry in the Planning-graph via Explanation-Guided Search. In *Proceedings of AAAI-99,* 1999.

Zimmerman, T. and Kambhampati, S. 2002. Using memory to transform Graphplan's search. Submitted to Eighteenth National Conference American on Artificial Intelligence, (AAAI 2002).

Zimmerman, T., Kambhampati, S. 2001. Effective Interplay of State Space and CSP Views in a Single Planner. ASU CSE TR-01-008.

Bonet, B. and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP-99,* 1999.

Nguyen, X. and Kambhampati, S., 2000. Extracting effective and admissible state space heuristics from the planning graph. In *Proc. AAAI-2000.*

Introducing Variable Importance Tradeoffs into CP-Nets

Ronen I. Brafman **Carmel Domshlak**
Ben-Gurion University
Beer-Sheva 84105, Israel
brafman,dcarmel@cs.bgu.ac.il

Abstract

The ability to make decisions and to assess potential courses of action is a corner-stone of many AI applications, and usually this requires explicit information about the decision-maker's preferences. In many applications, preference elicitation is a serious bottleneck. The user either does not have the time, the knowledge, or the expert support required to specify complex multi-attribute utility functions. In such cases, a method that is based on intuitive, yet expressive, preference statements is required. In this paper we suggest the use of TCP-nets, an enhancement of CP-nets, as a tool for representing, and reasoning about qualitative preference statements. We present and motivate this framework, define its semantics, and show how it can be used to perform constrained optimization.

Introduction

The ability to make decisions and to assess potential courses of action is a corner-stone of many AI applications, including expert systems, autonomous agents, decision-support systems, recommender systems, configuration software, and constrained optimization applications. To make good decisions, we must be able to assess and compare different alternatives. Sometimes, this comparison is performed implicitly, as in many recommender systems. However, in many cases explicit information about the decision-maker's preferences is required.

Utility functions are an ideal tool for representing and reasoning with preferences. However, they can be very difficult to elicit, and the effort required is not always possible or justified. Instead, one should resort to other, more qualitative forms of preference representation. Ideally, this qualitative information should be easily obtainable from the user by non-intrusive means. That is, we should be able to generate it from natural and relatively simple statements about preferences obtained from the user, and this elicitation process should be amenable to automation. In addition, automated reasoning with this representation should be feasible and efficient.

One relatively recent framework for preference representation that addresses these concerns is that of *Conditional Preference Networks* (CP-nets) (Boutilier *et al.* 1997; 1999). In CP-nets, the decision maker is asked to describe how her preference over the values of one variable depends on the value of other variables. For example, she may state that her preference for a dessert depends on the value of the main-course as well as whether or not she had an alcoholic beverage. Her choice of an alcoholic beverage depends on the main course and the time of day. This information is described by a graphical structure in which the nodes represent variables of interest and the edges represent dependence relations between the variables. Each node is annotated with a *conditional preference table* (CPT) describing the user's preference over alternative values of this node given different values of the parent nodes. CP-nets capture a class of intuitive and useful natural language statements of the form "I prefer the value x_0 for variable X given that $Y = y_0$ and $Z = z_0$". Such statements do not require complex introspection nor a quantitative assessment.

In (Boutilier *et al.* 1997) it was observed that there is another class of preferential statements, not captured by the CP-net model, that is no less intuitive or important. These statements have the following form: "It is more important to me that the value of X be high than that the value of Y be high." We call these *relative importance* statements. For instance, one might say "The length of the journey is more important to me than the choice of airline". A more refined notion of importance, though still intuitive and easy to communicate, is that of *conditional relative importance*: "The length of the journey is more important to me than the choice of airline provided that I am lecturing the following day. Otherwise, the choice of airline is more important." This latter statement is of the form: "A better assignment for X is more important than a better assignment for Y given that $Z = z_0$." Notice that information about relative importance is different from information about independence. In the example above, my preference for an airline does not depend on the duration of the journey because, e.g., I compare airlines based on their service, security levels and the quality of their frequent flyer program.

In this paper we show that enriching a CP-net based preferential relation by adding such statements may have a significant impact on both the consistency of the specified relation, and the reasoning about it. Likewise, we show that the internal structure of such a "mixed" preferential statement set can be exploited in order to achive efficiency in both consistency testing and in preferential reasoning. In particular,

we present an extension of CP-nets, which we call TCP-nets (for *tradeoffs-enhanced* CP-nets), and show how they can be used to compute optimal outcomes given constraints. TCP-nets capture both information about conditional independence and about conditional relative importance. Thus, they provide a richer framework for representing user preferences, allowing stronger conclusions to be drawn, yet remain committed to the use of intuitive, qualitative information as their source.

This paper is organized as follows. First, we describe the notions underlying TCP-nets: preference relations, preferential independence, and relative importance. Second, we define TCP-nets, and provide a number of examples. Third, we define the semantics of TCP-nets and discuss the conditions for the consistency of the specified preferential orders. Forth, we show how TCP-nets can be used to perform constrained optimization. We conclude with a discussion of future work. Proofs and a discussion of the TCP-nets applicability to the configuration problems appears in the full paper (Domshlak & Brafman 2002b).

Preference Orders, Independence, and Relative Importance

In this section we describe the ideas underlying TCP-nets: preference orders, preferential independence and conditional preferential independence, as well as relative importance and conditional relative importance.

Preference and Independence

A *preference relation* is a total pre-order (a *ranking*) over a set of outcomes. Given two outcomes o, o', we write $o \succeq o'$ to denote that o is at least as preferred as o' and we write $o \succ o'$ to denote that o is strictly more preferred than o'. The types of outcomes we are concerned with consist of possible assignments to some set of variables. More formally, we assume some given set $V = \{X_1, \ldots, X_n\}$ of variables with corresponding domains $\mathcal{D}(X_1), \ldots, \mathcal{D}(X_n)$. The set of possible outcomes is then $\mathcal{D}(X_1) \times \cdots \times \mathcal{D}(X_n)$. For example, in the context of the problem of configuring a personal computer (PC), the variables may be *processor type, screen size, operating system* etc., where *screen size* has the domain $\{$*17in, 19in, 21in*$\}$, *operating system* has the domain $\{$*LINUX, Windows98, WindowsXP*$\}$, etc. Each assignment to the set of variables specifies an outcome – a particular PC configuration. Thus, a preference ordering over these outcomes specifies a ranking over possible PC configurations.

The number of possible outcomes is exponential in n, while the set of possible total orders on them is doubly exponential in n. Therefore, explicit specification and representation of a ranking are not realistic. We must find implicit means of describing this preference relation. Often, the notion of preferential independence plays a key role in such representations. Intuitively, $\mathbf{X}$ and $\mathbf{Y} = \mathbf{V} - \mathbf{X}$ are *preferentially independent* if for all assignments to $\mathbf{Y}$, our preference over $\mathbf{X}$ values are identical. More formally, let $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{D}(\mathbf{X})$ for some $\mathbf{X} \subseteq \mathbf{V}$ (where we use $\mathcal{D}(\cdot)$ to denote the domain of a set of variables as well), and let $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{D}(\mathbf{Y})$, where $\mathbf{Y} = \mathbf{V} - \mathbf{X}$. We say that $\mathbf{X}$ is *preferentially independent* of $\mathbf{Y}$ iff, for all $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ we have that

$$\mathbf{x}_1\mathbf{y}_1 \succeq \mathbf{x}_2\mathbf{y}_1 \text{ iff } \mathbf{x}_1\mathbf{y}_2 \succeq \mathbf{x}_2\mathbf{y}_2$$

For example, in our PC configuration example, the user may assess *screen size* to be preferentially independent of *processor type* and *operating system*. This could be the case if the user always prefers a larger screen to a smaller screen, no matter what the processor or the OS are.

Preferential independence is a strong property, and therefore, less common. A more refined notion is that of conditional preferential independence. Intuitively, $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{Z}$ if for every fixed assignment to $\mathbf{Z}$, the ranking of $\mathbf{X}$ values is independent of the value of $\mathbf{Y}$. Formally, let $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ be a partition of $\mathbf{V}$ and let $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{z}$ iff, for all $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ we have that

$$\mathbf{x}_1\mathbf{y}_1\mathbf{z} \succeq \mathbf{x}_2\mathbf{y}_1\mathbf{z} \text{ iff } \mathbf{x}_1\mathbf{y}_2\mathbf{z} \succeq \mathbf{x}_2\mathbf{y}_2\mathbf{z}$$

$\mathbf{X}$ and $\mathbf{Y}$ are conditionally preferentially independent given $\mathbf{Z}$ if they are conditionally preferentially independent given any assignment $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. Returning to our PC example, the user may assess *operating system* to be independent of all other features given *processor type*. That is, it always prefers LINUX given an AMD processor and Windows98 given an Intel processor (e.g., because he might believe that Windows98 is optimized for the Intel processor, whereas LINUX is otherwise better). Note that the notions of preferential independence and conditional preferential independence are among a number of standard notions of independence in multi-attribute utility theory (Keeney & Raiffa 1976).

Relative Importance

Although statements of preferential independence are natural and useful, the orderings obtained by relying on them alone are relatively weak. To understand this, consider two preferentially independent boolean attributes A and B with values a_1, a_2 and b_1, b_2, respectively. If A and B are preferentially independent, then we can specify a preference order over A values, say $a_1 \succ a_2$, independently of the value of B. Similarly, our preference over B values, say $b_1 \succ b_2$, is independent of the value of A. From this we can deduce that a_1b_1 is the most preferred outcome and a_2b_2 is the least preferred outcome. However, we do not know the relative order of a_1b_2 and a_2b_1. This is typically the case when we consider independent variables: We can rank each one given a fixed value of the other, but often, we cannot compare outcomes in which both values are different. One type of information that can address some (though not necessarily all) such comparisons is information about relative importance. For instance, if we say that A is more important than B then this means that we prefer to reduce the value of B rather than reduce the value of A. In that case, we know that $a_1b_2 \succ a_2b_1$, and we can (totally) order the set of outcomes as $a_1b_1 \succ a_1b_2 \succ a_2b_1 \succ a_2b_2$.

Returning to our PC configuration example, suppose that *operating system* and *processor type* are independent at-

tributes. We might say that *processor type* is more important than *operating system*, e.g, because we believe that the effect of the processor's type on system performance is more significant than the effect of the operating system.

Formally, let a pair of variables X and Y be preferentially independent given $\mathbf{W} = \mathbf{V} - \{X, Y\}$. We say that X is *more important* than Y, denoted by $X \rhd Y$, if for every assignment $\mathbf{w} \in \mathcal{D}(\mathbf{W})$ and for every $x_i, x_j \in \mathcal{D}(X)$, $y_a, y_b \in \mathcal{D}(Y)$, such that $x_i \succ x_j$ given $\mathbf{w}$ and $y_b \succ y_a$ given $\mathbf{w}$, we have that:

$$x_i y_a \mathbf{w} \succ x_j y_b \mathbf{w}.$$

For instance, when both X and Y are binary variables, and $x_1 \succ x_2$ and $y_1 \succ y_2$ hold given $\mathbf{w}$, then $X \rhd Y$ iff we have $x_1 y_2 \mathbf{w} \succ x_2 y_1 \mathbf{w}$ for all $\mathbf{w} \in \mathcal{D}(\mathbf{W})$. Notice that this is a strict notion of importance – any reduction in Y is preferred to any reduction in X. Clearly, this idea can be refined by providing an actual ordering over elements of $\mathcal{D}(XY)$. We have decided not to pursue this option farther because it is less natural to specify. However, our results generalize to such specifications as well. In addition, one can consider relative importance assessments among more than two variables. However, we feel that such statements are somewhat artificial and less natural to articulate.

Relative importance information is a natural enhancement of independence information. It retains the property we value so much: it corresponds to statements that a naive user would find simple and clear to evaluate and articulate. Moreover, it can be generalized naturally to a notion of *conditional relative importance*. For instance, suppose that the relative importance of *processor type* and *operating system* depends on the primary usage of the PC. For example, when the PC is used primarily for graphical applications, then the choice of an operating system is more important than that of a processor because certain important software packages for graphic design are not available on LINUX. However, for other applications, the processor type is more important because applications for both Windows and LINUX exist. Thus, we say that X is more important than Y given $\mathbf{z}$ if we always prefer to reduce the value of Y rather than the value of X when $\mathbf{z}$ holds.

Formally, let $X, Y, \mathbf{W}$ be as above, and let $\mathbf{Z} \subseteq \mathbf{W}$. We say that X is *more important* than Y given an assignment $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$ (*ceteris paribus*) iff, for any assignment $\mathbf{w}$ on $\mathbf{W} = \mathbf{V} - (\{X, Y\} \cup \mathbf{Z})$ we have:

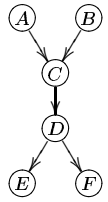
$$x_i y_a \mathbf{z}\mathbf{w} \succ x_j y_b \mathbf{z}\mathbf{w}$$

whenever $x_i \succ x_j$ given $\mathbf{z}\mathbf{w}$ and $y_b \succ y_a$ given $\mathbf{z}\mathbf{w}$. We denote this relation by $X \rhd_{\mathbf{z}} Y$. Finally, if for some $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$ we have that either $X \rhd_{\mathbf{z}} Y$, or $Y \rhd_{\mathbf{z}} X$, then we say that the relative importance of X and Y is conditioned on $\mathbf{Z}$, and write $\mathcal{RI}(X, Y, \mathbf{Z})$.

TCP Nets

TCP-nets (for *CP-nets with tradeoffs*) is an extension of CP-nets (Boutilier *et al.* 1999) that encodes (conditional) preferential independence and (conditional) relative importance statements. We use this graph-based representation for two reasons: First, it is an intuitive visual representation of preference independence and relative importance statements. Second, the structure of the graph has important consequences to issues such as consistency and complexity of reasoning. For instance, one of the basic results we present later shows that when this structure is "acyclic" (for a suitable definition of this notion!), then the preference statements contained in the graph are consistent – that is, there is a total pre-order that satisfies them.

TCP-nets are annotated graphs with three types of edges. The nodes of a TCP-net correspond to the problem variables $\mathbf{V}$. The first type of (directed) edge captures preferential dependence, i.e., an edge from X to Y implies that the user has different preferences over X values given different values of Y. The second (directed) edge type captures relative importance relations. The existence of such an edge from X to Y implies that X is more important than Y. The third (undirected) edge type captures conditional importance relations: Such an edge between nodes X and Y exists if there exists some $\mathbf{Z}$ for which $\mathcal{RI}(X, Y, \mathbf{Z})$ holds.



$[a_1 \succ a_2]\ [b_2 \succ b_1]$
$[(a_1 \wedge b_1) \vee (a_2 \wedge b_2) : c_1 \succ c_2;$
$(a_1 \wedge b_2) \vee (a_2 \wedge b_1) : c_2 \succ c_1]$
$[c_1 : d_1 \succ d_2;\ c_2 : d_2 \succ d_1]$
$[d_1 : e_1 \succ e_2;\ d_2 : e_2 \succ e_1]$
$[d_1 : f_1 \succ f_2 \succ f_3;\ d_2 : f_3 \succ f_1 \succ f_2]$

Figure 1: An example CP-net

Like in CP-nets, each node X in a TCP-net is annotated with a *conditional preference table* (CPT). This table associates a preferences over $\mathcal{D}(X)$ for every possible value assignment to the parents of X (denoted $Pa(X)$). In addition, in TCP-nets, each undirected edge is annotated with a *conditional importance table* (CIT). The CIT associated with such an edge (X, Y) describes the relative importance of X and Y given the value of the conditioning variables.

Formally, a TCP-net $\mathcal{N}$ is a tuple $\langle \mathbf{V}, \mathsf{cp}, \mathsf{i}, \mathsf{ci}, \mathsf{cpt}, \mathsf{cit} \rangle$:

1. $\mathbf{V}$ is a set of nodes, corresponding to the problem variables $\{X_1, \ldots, X_n\}$.
2. cp is a set of directed cp-*arcs* $\{\alpha_1, \ldots, \alpha_k\}$ (where cp stands for *conditional preference*). A cp-arc $\langle \overrightarrow{X_i, X_j} \rangle$ belongs to $\mathcal{N}$ iff the preferences over the values of X_j depend on the actual value of X_i.
3. i is a set of directed i-*arcs* $\{\beta_1, \ldots, \beta_l\}$ (where i stands for *importance*). An i-arc $(\overrightarrow{X_i, X_j})$ belongs to $\mathcal{N}$ iff $X_i \rhd X_j$.
4. ci is a set of undirected ci-*arcs* $\{\gamma_1, \ldots, \gamma_m\}$ (where ci stands for *conditional importance*). A ci-arc (X_i, X_j) belongs to $\mathcal{N}$ iff we have $\mathcal{RI}(X_i, X_j, \mathbf{Z})$ for some $\mathbf{Z} \subseteq \mathbf{V} - \{X_i, X_j\}$.
5. cpt associates a CPT with every node $X \in \mathbf{V}$. A CPT is from $\mathcal{D}(Pa(X))$ (i.e., assignment's to X's parent nodes) to total pre-orders over $\mathcal{D}(X)$.
6. cit associates with every ci-arc (X_i, X_j) a subset $\mathbf{Z}$ of $\mathbf{V} - \{X_i, X_j\}$ and a mapping from a subset of $\mathcal{D}(\mathbf{Z})$ to total

orders over the set $\{X_i, X_j\}$. We call $\mathbf{Z}$ the *selector set* of (X_i, X_j) and denote it by $\mathcal{S}(X_i, X_j)$.[1]

A CP-net (Boutilier *et al.* 1999) is simply a TCP-net in which the sets i and ci (and therefore cit) are empty, and that every node $X \in \mathbf{V}$ is independent of all other nodes given $Pa(X)$. In the rest of this section we provide examples of TCP-net. We start with an example of a CP-net shown in Figure 1.

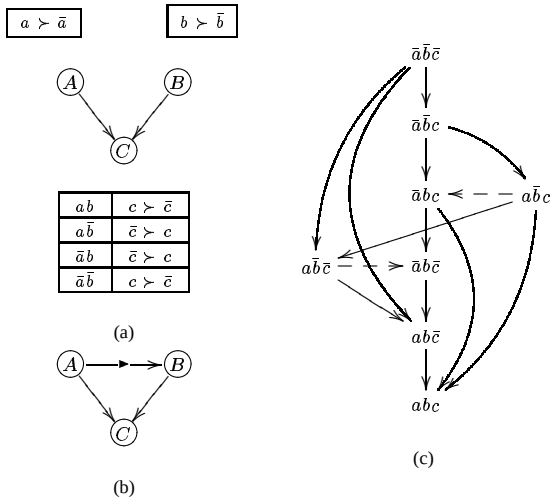


Figure 2: Illustrations for Example 2.

Example 1 The CP-net in Figure 1 is defined over the variables $\{A, B, C, D, E, F\}$; all variables are binary except for the three-valued F. The decision maker specifies unconditional preference over the values of a (denoted in figure by $a_1 \succ a_2$). However, if $A = a_1$ and $B = b_2$ the decision maker prefers c_2 to c_1 (denoted by $(a_1 \wedge b_2) : c_2 \succ c_1$). Now consider this CP-net and the following three outcomes: $\alpha = a_1b_1c_1d_2e_2f_2$, $\beta = a_1b_1c_2d_2e_2f_2$, and $\gamma = a_1b_1c_2d_1e_2f_2$. α and β assign the same values to all variables except C. α assigns to C a value that is preferred to the value β assigns to C, given the assignment to the parents of C (denoted $Pa(C)$). Therefore, $\alpha \succ \beta$ is a consequence of this CP-net. The same argument applies to β and γ, with respect to the variable D, and thus, $\beta \succ \gamma$ is a consequence of this CP-net as well. $\alpha \succ \gamma$ cannot be derived directly from the CP-net above. However, this relation can be inferred via transitivity from $\alpha \succ \beta$ and $\beta \succ \gamma$.

In the following examples all variables are binary, although the semantics of both CP-net and TCP-net is defined with respect to arbitrary finite domains.

Example 2 Figure 2(a) illustrates a simple CP-net over three variables A, B, and C: a is unconditionally preferred to $\bar{a}$, and b is unconditional preferred to $\bar{b}$, while the preference over the values of C is conditioned on both A and B. The solid lines in Figure 2(c) show the preference relation that this CP-net induces. The top and the bottom elements are the worst and the best outcomes, respectively. Arrows are directed from less preferred to more preferred outcomes. In turn, Figure 2(b) displays a TCP-net that extends this CP-net by adding an i-arc from A to B. Thus, A is absolutely more important than B. This induces additional relations among outcomes, captured by the dashed lines in Figure 2(c).

[1]Naturally we expect this set $\mathbf{Z}$ to be the minimal context upon which the relative importance between X_i and X_j depends.

Example 3 Figure 3(a) illustrates a CP-net over five variables A, B, C, D, and E. Figure 3(b) presents a TCP-net that extends this CP-net by adding an i-arc from B to E and a ci-arc between C and D. The relative importance of C and D depends on the assignment to B and E. When B and E are assigned be, then $C \rhd D$. When B and E are assigned $b\bar{e}$ or $\bar{b}e$, then $D \rhd C$. Finally, when B and E are assigned $\bar{b}\bar{e}$, the relative importance between C and D is unspecified. The CIT of this ci-arc is also presented in Figure 3(b).

Semantics and Consistency

The semantics of a TCP-net is straightforward, and is defined in terms of the set of preference rankings that are consistent with the set of constraints imposed by its preference and importance information. We use $\succ^X_{\mathbf{u}}$ to denote the preference relation over the values of X given an assignment $\mathbf{u}$ to $\mathbf{U} \supseteq Pa(X)$.

Definition 1 Let $\mathcal{N}$ be a TCP-net over a set of variables $\mathbf{V}$.

1. Let $\mathbf{W} = \mathbf{V} - \{X\} \cup Pa(X)$ and let $\mathbf{p} \in \mathcal{D}(Pa(X))$. A preference ranking $\succ$ satisfies $\succ^X_{\mathbf{p}}$ iff $x_i\mathbf{pw} \succ x_j\mathbf{pw}$, for each $\mathbf{w} \in \mathcal{D}(\mathbf{W})$, when $x_i \succ^X_{\mathbf{p}} x_j$ holds.
2. A preference ranking $\succ$ satisfies the CPT of X iff it satisfies $\succ^X_{\mathbf{p}}$ for each assignment $\mathbf{p}$ of $Pa(X)$.
3. A preference ranking $\succ$ satisfies $X \rhd Y$ iff for every $\mathbf{w} \in \mathcal{D}(\mathbf{W})$ s.t. $\mathbf{W} = \mathbf{V} - \{X, Y\}$, $x_iy_a\mathbf{w} \prec x_jy_b\mathbf{w}$ whenever $x_i \succ^X_{\mathbf{w}} x_j$ and $y_b \succ^Y_{\mathbf{w}} y_a$.
4. A preference ranking $\succ$ satisfies $X \rhd_{\mathbf{z}} Y$ iff for every $\mathbf{w} \in \mathcal{D}(\mathbf{W})$ s.t. $\mathbf{W} = \mathbf{V} - \{X, Y\} \cup \mathbf{Z}$, $x_iy_a\mathbf{zw} \prec x_jy_b\mathbf{zw}$ whenever $x_i \succ^X_{\mathbf{zw}} x_j$ and $y_b \succ^Y_{\mathbf{zw}} y_a$.
5. A preference ranking $\succ$ satisfies the CIT of the ci-arc (X, Y) if it satisfies $X \rhd_{\mathbf{z}} Y$ whenever an entry in the table conditioned of $\mathbf{z}$ ranks X as more important.

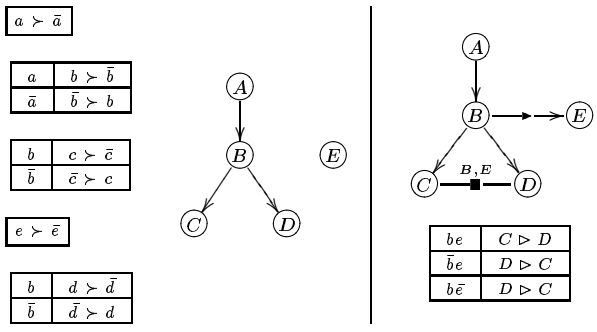


Figure 3: Illustrations for Example 3.

A preference ranking $\succ$ satisfies a TCP-net $\mathcal{N}$ iff it: (i) satisfies every CPT in $\mathcal{N}$; (ii) satisfies $X \rhd Y$ for every i-arc (X_i, X_j) in $\mathcal{N}$; (iii) satisfied every CIT in $\mathcal{N}$. A TCP-net is *satisfiable* iff there is some ranking $\succ$ that satisfies it. Finally, $o \succ o'$ is *implied* by a TCP-net iff it holds in all preference rankings that satisfy this TCP-net.

Lemma 1 (Transitivity) *If $o \succ o'$ and $o' \succ o''$ are implied by a TCP-net, then so is $o \succ o''$.*

We now define two types of directed graphs that are induced by a TCP-net $\mathcal{N}$.

Definition 2 $\mathcal{N}$'s *dependency graph* contains all nodes and directed edges of $\mathcal{N}$ (i.e., the cp-arcs and the i-arcs)) as well as the edges (X_k, X_i) and (X_k, X_j) for every ci-arc (X_i, X_j) in $\mathcal{N}$ and every $X_k \in \mathcal{S}(X_i, X_j)$.

Let $\mathcal{S}(\mathcal{N})$ be the union of all selector sets of $\mathcal{N}$. Given an assignment $\mathbf{w}$ to $\mathcal{S}(\mathcal{N})$, the $\mathbf{w}$-*directed* graph of $\mathcal{N}$ contains all nodes and directed edges of $\mathcal{N}$ and the edge from X_i to X_j if (X_i, X_j) is a ci-arc of $\mathcal{N}$ and the CIT for (X_i, X_j) specifies that $X_i \rhd X_j$ given $\mathbf{w}$.

Definition 3 A TCP-net $\mathcal{N}$ is *conditionally acyclic* if its induced dependency graph is acyclic and for every assignment $\mathbf{w}$ to $\mathcal{S}(\mathcal{N})$, the induced $\mathbf{w}$-directed graphs are acyclic.

Theorem 1 *Every conditionally acyclic TCP-net is satisfiable.*

Verifying conditional acyclicity requires verifying two properties. The verification of acyclicity of the dependency graph is simple. Naive verification of the acyclicity of every $\mathbf{w}$-directed graph can require time exponential in the combined size of the selector sets. Following we show some sufficient and/or neccessary conditions for the $\mathbf{w}$-directed graphs acyclicity that are much easier to check.

Let $\mathcal{N}$ be a TCP-net. If $\mathcal{N}$ contains directed cycles, then surely both the induced dependency graph and every $\mathbf{w}$-directed graph is cyclic. Since such directed cycles are simple to detect, let us assume that they do not arise in $\mathcal{N}$. Next, note that if there are no cycles in the undirected graph induced by $\mathcal{N}$ (i.e., the graph obtained from $\mathcal{N}$ by removing the direction of its directed edges) then clearly all $\mathbf{w}$-directed graphs are acyclic. Again, this case too is quite simple to check. Finally, if there are undirected cycles, but each such cycle, when projected back to $\mathcal{N}$, contains directed arcs in different directions, then all $\mathbf{w}$-directed graphs are still acyclic. This latter sufficient condition can be checked in (low) polynomial time.

Thus, we are left with the situation that $\mathcal{N}$ contains sets $\mathcal{A}$ of edges that form a cycle in the induced undirected graph, not all of these edges are directed, yet all the directed edges point in the same direction (i.e., clockwise or counterclockwise). We call these sets *semi-directed* cycles, and we concentrate on their investigation in the rest of this section.

Each assignment $\mathbf{w}$ to the selector sets of ci-arcs in a semi-directed cycle $\mathcal{A}$ induces a direction to all these arcs. We say that $\mathcal{A}$ is *conditionally acyclic* if under no such assignment $\mathbf{w}$ do we obtain a directed cycle from $\mathcal{A}$. Otherwise, $\mathcal{A}$ is *conditionally directed*. Our first observation is that if all semi-directed cycles in $\mathcal{N}$ are conditionally acyclic, then so is $\mathcal{N}$. Let $\mathcal{S}(\mathcal{A})$ be the union of the selector sets of all ci-arcs in $\mathcal{A}$. The time required to check for the conditional acyclicity of a semi-directed cycle $\mathcal{A}$ is exponential in the size of $\mathcal{S}(\mathcal{A})$. Thus, if $\mathcal{S}(\mathcal{A})$ is small for each semi-directed cycle $\mathcal{A}$ in the network, then conditionally acyclicity can be checked for quickly. In fact, often we can determine that a semi-directed cycle is conditionally directed/acyclic even more efficiently.

Lemma 2 *Let $\mathcal{A}$ be a semi-directed cycle in $\mathcal{N}$. If $\mathcal{A}$ is conditionally acylic then it contains a pair of ci-arcs γ_i, γ_j such that $\mathcal{S}(\gamma_i) \cap \mathcal{S}(\gamma_j) \neq \emptyset$.*

In other words, if the selector sets of the ci-arcs in $\mathcal{A}$ are all pairwise disjoint, then $\mathcal{A}$ is conditionally directed. Thus, Lemma 2 provides a necessary condition for conditional acyclicity of $\mathcal{A}$ that can be checked in time polynomial in the number of variables.

Lemma 3 *$\mathcal{A}$ is conditionally acyclic if it contains a pair of ci-arcs γ_i, γ_j such that either:*

(a) $\mathcal{A}$ contains directed edges and for each assignment $\mathbf{w}$ to $\mathcal{S}(\gamma_i) \cap \mathcal{S}(\gamma_j)$, γ_i or γ_j can be converted into an i-arc that violates the direction of the directed edges of $\mathcal{A}$.

(b) All edges in $\mathcal{A}$ are undirected and for each assignment $\mathbf{w}$ to $\mathcal{S}(\gamma_i) \cap \mathcal{S}(\gamma_j)$, γ_i and γ_j can be converted into i-arcs that point in opposite directions w.r.t. $\mathcal{A}$.

Lemma 3 provides a sufficient condition for conditional acyclicity of $\mathcal{A}$ that can be checked in time exponential in the maximal size of selector set intersection for a pair of ci-arcs in $\mathcal{A}$. Note that the TCP-net size is at least of this complexity (because of the CITs description), thus checking this condition is only linear in the size of the network.

Lemma 4 *Let $shared(\mathcal{A})$ be the union of all pairwise intersections of the selector sets of the ci-arcs in $\mathcal{A}$:*

$$shared(\mathcal{A}') = \bigcup_{\gamma_i, \gamma_j \in \mathcal{A}} \mathcal{S}(\gamma_i) \cap \mathcal{S}(\gamma_j)$$

If $\mathcal{A}$ contains some cp or i arcs, then $\mathcal{A}$ is conditionally acyclic if and only if, for each assignment π on $shared(\mathcal{A})$, there exists a ci-arc $\gamma_\pi \in \mathcal{A}$ that, given π, can be converted into an i-arc that violates the direction of $\mathcal{A}$.

Otherwise, if $\mathcal{A}$ consists only of ci-arcs, then $\mathcal{A}$ is conditionally acyclic if and only if, for each assignment π on $shared(\mathcal{A})$, there exist two ci-arcs $\gamma_\pi^1, \gamma_\pi^2 \in \mathcal{A}'$ that, given π, can be converted into i-arcs that disagree on the direction with respect to $\mathcal{A}$.

In general, the size of $shared(\mathcal{A})$ is $O(|\mathbf{V}|)$, thus checking the (necessary and sufficient) condition provided by Lemma 4 is generally hard. However, it is clear that $|shared(\mathcal{A})| \leq |\mathcal{S}(\mathcal{A})|$. Therefore, checking this condition

is more efficient than checking the naive one. Likewise, restricting the size of $shared(\mathcal{A})$ (in order to ensure polynomial time consistency verification) will leave us with a much richer set of TCP-nets than restricting the size of $\mathcal{S}(\mathcal{A})$.

Preferential Constraint-based Optimization

One of the central properties of the original CP-net model that was presented in (Boutilier *et al.* 1999) is that, given an acyclic CP-net $\mathcal{N}$ and a partial assignment π on its variables, it is simple to determine an outcome consistent with π that is *preferentially optimal* with respect to $\mathcal{N}$. The corresponding procedure is as follows: Traverse the variables in some topological order induced by $\mathcal{N}$ and set each unassigned variable to its most preferred value given its parents' values. Our immediate observation is that this procedure works *as is* also for conditionally acyclic TCP-nets: The relative importance relations do not play a role in this case, and the network is traversed according to a topological order induced by the CP-net part of the given TCP-net.

This strong property of optimization with respect to the acyclic CP-nets (and the conditionally acyclic TCP-nets) does not hold if some of the TCP-net variables are mutually constrained by a set of hard constraints, $\mathcal{C}$. In this case, determining the set of Pareto-optimal[2] feasible outcomes is not trivial. For the acyclic CP-nets, a branch and bound algorithm for determining the optimal feasible outcomes was introduced in (Boutilier *et al.* 1997). This algorithm has the important *anytime* property – once an outcome is added to the current set of non-dominated outcomes, it is never removed. In this algorithm, variables are instantiated according to a topological ordering. Thus, more important variables, i.e., variables that are "higher-up" in the network, are assigned values first.

Figure 4 presents our extension/modification of that algorithm to conditionally acyclic TCP-nets which retains the *anytime* property. The key difference between processing acyclic CP-net and conditionally acyclic TCP-net is that latter induces a *set* of partial orderings, corresponding to different assignments on its selector variables. Consider a conditionally acyclic TCP-net $\mathcal{N}$. The set of partial orders induced by $\mathcal{N}$ over its variables is consistent with the dependency graph of $\mathcal{N}$. In addition, if $\mathcal{S}(\mathcal{N})$ is the union of the selector variables in $\mathcal{N}$, then let $\mathcal{S}'(\mathcal{N}) \subseteq \mathcal{S}(\mathcal{N})$ be a *prefix* of $\mathcal{S}(\mathcal{N})$ if and only if, for each $X \in \mathcal{S}'(\mathcal{N})$, and for each $Y \in \mathcal{S}(\mathcal{N}) \setminus \mathcal{S}'(\mathcal{N})$, X is not reachable from Y in the dependency graph of $\mathcal{N}$. Observe, that any set of partial orders over the variables of $\mathcal{N}$, that agree on an assignment on a prefix $\mathcal{S}'(\mathcal{N})$ of $\mathcal{S}(\mathcal{N})$, agree on ordering of all the variables in $\mathcal{N}$, relative importance of which is fully determined by $\mathcal{S}'(\mathcal{N})$.

The Search algorithm is guided by the underlying TCP-net $\mathcal{N}$. It proceeds by assigning values to the variables in a top-down manner, assuring that outcomes are generated according to the preferential ordering induced by $\mathcal{N}$ – on a call to the Search procedure with a TCP-net $\mathcal{N}$, the eliminated variable X is one of the root variables of $\mathcal{N}$. The values of X are considered according to the preferential ordering induced by the assignment on $Pa(X)$. Note that X is observed in some context $\mathcal{K}$ which necessarily contains some assignment on $Pa(X)$. Whenever a variable X is assigned to a value x_i, the current set of constraints $\mathcal{C}$ is being strengthened into $\mathcal{C}_i$. As a result of this propagation of $X = x_i$, values for some variables (at least for the variable X) will be fixed automatically, and this partial assignment $\mathcal{K}'$ will extend the current context $\mathcal{K}$ in processing of the next variable. The Reduce procedure refines the TCP-net $\mathcal{N}$ with respect to $\mathcal{K}'$: For each variable assigned by $\mathcal{K}'$, we reduce both the CPTs and the CITs involving this variable, and remove this variable from the network. This reduction of the CITs may remove conditioning of relative importance between some variables, and thus convert some ci-arcs into i-arcs, and/or to remove some ci-arcs completely. The central point is that, in contrast to CP-nets, for a pair X values x_i, x_j, the dependency graphs of the networks $\mathcal{N}_i$ and $\mathcal{N}_j$,

Search ($\mathcal{N}$, $\mathcal{C}$, $\mathcal{K}$)
Input: Conditionally acyclic TCP-net $\mathcal{N}$, Constraints $\mathcal{C}$,
Context $\mathcal{K}$ (partial assignment on the original TCP-net)
Output: Set of all, Pareto-optimal w.r.t. $\mathcal{N}$, solutions for $\mathcal{C}$.

Choose any variable X s.t. there is no cp-arc $\langle \overrightarrow{Y, X} \rangle$,
no i-arc $(\overrightarrow{Y, X})$, and no (X, Y) in $\mathcal{N}$.
Let $x_1 \succ \ldots \succ x_k$ be the preference ordering of $\mathcal{D}(X)$
given the assignment on $Pa(X)$ in $\mathcal{K}$.
Initialize the set of local results by $\mathcal{R} = \emptyset$
for $(i = 1;\ i \leq k;\ i++)$ **do**
  $X = x_i$
  Strengthen the constraints $\mathcal{C}$ by $X = x_i$ to obtain $\mathcal{C}_i$
  if $\mathcal{C}_j \subseteq \mathcal{C}_i$ for some $j < i$ **or** $\mathcal{C}_i$ is inconsistent **then**
    continue with the next iteration
  else
    Let $\mathcal{K}'$ be the partial assignment induced by $X = x_i$ and $\mathcal{C}_i$
    $\mathcal{N}_i$ = Reduce ($\mathcal{N}$,$\mathcal{K}'$)
    Let $\mathcal{N}_i^1, \ldots, \mathcal{N}_i^m$ be the components of $\mathcal{N}_i$, connected
      either by the edges of $\mathcal{N}_i$ or by the constraints $\mathcal{C}_i$.
    for $(j = 1;\ j \leq m;\ j++)$ **do**
      $\mathcal{R}_i^j$ = Search ($\mathcal{N}_i^j, \mathcal{K} \cup \mathcal{K}', \mathcal{C}_i$)
    if $\mathcal{R}_i^j \neq \emptyset$ for all $j \leq m$ **then**
      foreach $o \in \mathcal{K}' \times \mathcal{R}_i^1 \times \cdots \times \mathcal{R}_i^m$ **do**
        if for each $o' \in \mathcal{R}$ holds $\mathcal{K} \cdot o' \not\succ \mathcal{K} \cdot o$ **then** Add o to $\mathcal{R}$
return $\mathcal{R}$
Reduce ($\mathcal{N}$, $\mathcal{K}'$)
foreach $\{X = x_i\} \in \mathcal{K}'$ **do**
  foreach cp-arc $\langle \overrightarrow{X, Y} \rangle \in \mathcal{N}$ **do**
    Restrict the CPT of Y to the rows dictated by $X = x_i$.
  foreach ci-arc $\gamma = (Y_1, Y_2) \in \mathcal{N}$ s.t. $X \in \mathcal{S}(\gamma)$ **do**
    Restrict the CIT of γ to the rows dictated by $X = x_i$.
    if, given the restricted CIT of γ, relative importance
      between Y_1 and Y_2 is independent of $\mathcal{S}(\gamma)$, **then**
      if CIT of γ is not empty **then**
        Replace γ by the corresponding i-arc.
      else Remove γ.
  Remove from $\mathcal{N}$ all the edges involving X. **return** $\mathcal{N}$.

Figure 4: The Search algorithm for TCP-nets.

[2] An outcome o is said to be Pareto-optimal with respect to some preference order $\succ$ and a set of outcomes S if there is no other o' such that $o' \succ o$.

accepted by propagating $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively, may *disagree* on the ordering of some variables.

If the partial assignment $\mathcal{K}'$ causes the current CP-net to become disconnected with respect to both the edges of the network and the inter-variable constraints, then each connected component invokes an independent search. This is because optimization of the variables within such a component is independent of the variables outside that component. In addition, after strengthening the set of constraints $\mathcal{C}$ by $X = x_i$ to $\mathcal{C}_i$, some pruning is taking place in the search tree (see the **continue** instruction in the algorithm).[3] Therefore, the search is depth-first branch-and-bound, where the set of nondominated solutions generated so far is a proper subset of the required set of the Pareto-optimal solutions for the problem, and thus it corresponds to the current lower bound.

When the potentially nondominated solutions for a particular subgraph are returned with some assignment $X = x_i$, each such solution is compared to all nondominated solutions involving more preferred (in the current context $\mathcal{K}$) assignments $X = x_j$, $j < i$. A solution with $X = x_i$ is added to the set of the nondominated solutions for the current subgraph and context if and only if it passes this nondomination test. Note that, from the semantics of the TCP-net, given the same context $\mathcal{K}$, a solution involving $X = x_i$ can not be preferred to a solution involving $X = x_j$, $j < i$. Thus, the generated global set $\mathcal{R}$ never shrinks.

If we are interested in getting *one* Pareto-optimal solution for the given set of constraints (which is usually the case), then we can output the *first* feasible outcome that is generated by Search. No dominance queries between pairs of outcomes are required because there is nothing to compare the first accepted solution with. If we are interested in getting *all*, or even *a few* Pareto-optimal solutions, then the efficiency of the dominance queries becomes an important factor in the entire complexity of the Search algorithm.

The dominance query for a pair of outcomes can be stated as follows: Given a TCP-net $\mathcal{N}$ and two outcomes a and b, is $a \succ b$ a consequence of $\mathcal{N}$? In (Boutilier *et al.* 1999) this inference problem with respect to the CP-nets was treated as a search for a *flipping sequence* from the (purported) less preferred outcome b to the (purported) more preferred outcome b through a sequence of more preferred outcomes:

$$b = c_0 \prec c_1 \prec \cdots \prec c_{m-1} \prec c_m = a$$

where, for $0 \leq i < m$, outcome c_i is different from the outcome c_{i+1} in the value of exactly one variable X_j, and $c_i[j] \prec c_{i+1}[j]$ given the (same) values of $Pa(X_j)$ in c_i and c_{i+1}. Thus, each value flip in such a flipping sequence is sanctioned by the CP-net $\mathcal{N}$. The complexity of the search for a flipping sequence was analysed in (Domshlak & Brafman 2002a), and both polynomial and hard cases were presented with respect to the form of the CP-net.

The dominance inference problem with respect to the TCP-nets can be also treated as a search for an improving flipping sequence, where the notion of flipping sequence is extended from this for the CP-nets.

[3]This pruning was introduced in (Boutilier *et al.* 1997). See (Boutilier *et al.* 1997) for its explanation and justification.

Definition 4 A sequence of outcomes

$$b = c_0 \prec c_1 \prec \cdots \prec c_{m-1} \prec c_m = a$$

is an *improving flipping sequence with respect to a TCP-net* $\mathcal{N}$ is and only if, for $0 \leq i < m$, either

1. *(CP-flips)* outcome c_i is different from the outcome c_{i+1} in the value of exactly one variable X_j, and $c_i[j] \prec c_{i+1}[j]$ given the (same) values of $Pa(X_j)$ in c_i and c_{i+1}, or
2. *(I-flips)* outcome c_i is different from the outcome c_{i+1} in the value of exactly *two* variables X_j and X_k, $c_i[j] \prec c_{i+1}[j]$ and $c_i[k] \succ c_{i+1}[k]$ given the (same) values of $Pa(X_j)$ and $Pa(X_k)$ in c_i and c_{i+1}, and $X_j \rhd X_k$ given $\mathcal{RI}(X_j, X_k, \mathbf{Z})$ and the (same) values of $\mathbf{Z}$ in c_i and c_{i+1}.

Clearly, each value flip in such a flipping sequence is sanctioned by the TCP-net $\mathcal{N}$, and the CP-flips are exactly the flips allowed in CP-nets.

Lemma 5 *Given a TCP-net $\mathcal{N}$, and two outcomes a and b, $a \succ b$ is a consequence of $\mathcal{N}$ if and only if there is an improving flipping sequence from b to a with respect to $\mathcal{N}$.*

In general, various methods can be used for search for the flipping sequences. In particular, we believe that at least some of the techniques, developed for this task with respect to CP-nets in (Boutilier *et al.* 1999; Domshlak & Brafman 2002a) can be extended for the TCP-net model. However, complexity analysis of dominance testing for TCP-nets is not in the scope of this paper, and we leave it as an open problem for further research.

Conclusions

In this paper we introduced the notions of absolute and conditional *relative importance* between pairs of variables and extended the CP-net model (Boutilier *et al.* 1999) to capture these preference statements. The extended model is called TCP-net. We identified a wide class of TCP-nets that are satisfiable – the class of conditionally acyclic TCP-nets. Finally, we showed how this subclass of TCP-nets can be used in preference-based constrained optimization. We refer the reader to the full version of this paper, where the relevance of the TCP-net model to the area of product configuration is discussed.

An important open theoretical question is the precise complexity of dominance testing in TCP-nets (i.e., the question of which of two outcomes is more preferred). In the context of CP-nets, this problem was recently analyzed in (Domshlak & Brafman 2002a). Although the results from (Domshlak & Brafman 2002a) seem not to be immediately adaptable to TCP-nets, we believe that a corresponding, perhaps completely different, computational analysis is possible for TCP-nets. Finally, the question of consistency of TCP-nets that are not conditionally acyclic is another important challenge.

One of the areas in which we see significant potential for TCP-nets is *automatic configuration* (Sabin & Weigel 1998). While there has been a wide and growing body of research

on modeling and solving configuration problem, there is still a need for more work on modeling and learning user preferences, and using these to achieve configurations that are not only feasible, but also satisfactory from the user's point of view. This goal is emphasized by almost every paper on configuration, e.g. (Freuder & O'Sullivan 2001; Haag 1998; Junker 2001), especially in the context of high-level configurators (Haag 1998) for real-life domains.

Another interesting issue is the ability to acquire qualitative preference models from speech/text in natural language (James 1999). The intuitiveness of the qualitative preferential statements of TCP-net is closely related to the fact that they have a straightforward representation in everyday natural language. In addition, the corresponding preferential statements in a natural language seems to form a domain that is apriori constrained in a very special manner, and where specialized natural language techniques could apply.

References

Boutilier, C.; Brafman, R.; Geib, C.; and Poole, D. 1997. A Constraint-Based Approach to Preference Elicitation and Decision Making. In *AAAI Spring Symposium on Qualitative Decision Theory*.

Boutilier, C.; Brafman, R.; Hoos, H.; and Poole, D. 1999. Reasoning with Conditional Ceteris Paribus Preference Statements. In *Proc. of UAI-99*, 71–80.

Domshlak, C., and Brafman, R. 2002a. CP-nets - Reasoning and Consistency Testing. In *to appear in Proc. of KR-02*.

Domshlak, C., and Brafman, R. 2002b. Representing and Reasoning with Qualitative Statements Using TCP-nets. Technical Report CS-02-03, Dept. of Computer Science, Ben-Gurion Univ.

Freuder, E., and O'Sullivan, B. 2001. Modeling and Generating Tradeoffs for Constraint-Based Configuration. In *Workshop on Configuration (IJCAI-01)*.

Haag, A. 1998. Sales Configuration in Business Processes. *IEEE Intelligent Systems and their Applications* 13(4):78–85.

James, G. 1999. Challenges for Spoken Dialogue Systems. In *Proc. of IEEE ASRU Workshop*.

Junker, U. 2001. Preference Programming for Configuration. In *Workshop on Configuration (IJCAI-01)*, 50–56.

Keeney, R. L., and Raiffa, H. 1976. *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley.

Sabin, D., and Weigel, R. 1998. Product Conguration Frameworks - A Survey. *IEEE Intelligent Systems and their Applications* 13(4):42–49.