Skyline and Top-k Processing in Web Bargaining

Mohamed A. Soliman[1] Ihab F. Ilyas[1]
Nick Koudas[2]
[1]School of Computer Science, University of Waterloo, Canada
{m2ali,ilyas}@uwaterloo.ca
[2] Department of Computer Science, University of Toronto
koudas@cs.toronto.edu

University of Waterloo
Technical Report CS-2006-45

Abstract

Skyline and top-k queries are gaining increasing importance in many emerging applications. Current skyline and top-k query processing techniques work on deterministic object attributes and known scores. However, in many practical scenarios these settings are inapplicable. In this paper we focus on web interaction scenarios where each interaction is a data object with a set of possible outcomes (scores). Obtaining exact interaction scores is expensive as it involves complex arbitration between interacting parties. Moreover, the outcome of each interaction might redefine other interactions scores. We demonstrate that the search space for solving such problems is very large. Based on this we formulate and present skyline and top-k processing algorithms that can efficiently reduce the search space. We present the results of a thorough experimental evaluation quantifying the relative performance of the algorithms we propose herein with respect to costly exact solutions. Our results indicate that our techniques can efficiently reduce the space and identify precise solutions.

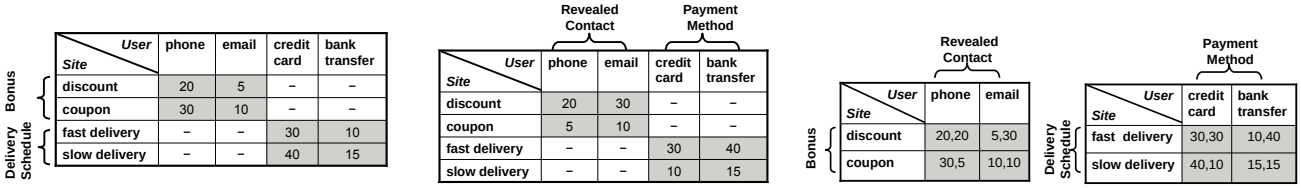
1 Introduction

Skyline (pareto-optimal) objects over d dimensions is the set of data objects that are not dominated by any other objects restricted to those dimensions. On the other hand, top–k queries return the k objects with the highest scores according to some scoring function. Skyline and top–k queries are gaining increasing importance in relational databases [11, 15], multimedia search [4], preference queries [10, 13], and data mining [23]. Current techniques assume deterministic settings and functions that return exact scores. However, applications in e-commerce, probabilistic databases and sensor networks often deal with various sources of uncertainty. Factoring data uncertainty in skyline and top-k processing is lacking in the current approaches.

In this paper we initiate work in the direction of skyline and top-k processing in bargaining and web interaction scenarios, where two parties interact in order to implement a transaction that eventually yields mutually beneficial outcome. In contrast to typical applications, the data objects we consider in these settings are regions of possible scores.

Several negotiation systems, e.g. INSPIRE [1], SimpleNS [2], SmartSettle [3], are currently available to facilitate negotiation and preference match-making using trusted centralized servers. Such systems help web users find competitive bargains. Automated agents that do negotiation on behalf of users or business parties is an interesting line of ongoing research. The identification of interesting bargains to all interacting parties is challenging in these settings.

The ability of interacting parties to negotiate has to be supported by an underlying platform to express and manage preferences. For example consider the case of the Platform for Privacy Preferences (P3P) protocol [8]. It is an XML based specification for users (consumers) and sites (businesses) to declare their privacy preferences. A user can declare, for example, that she is willing to reveal an email address but not a telephone number. In a similar fashion a business can declare types of contacts, payments, etc that are acceptable. Custom pieces of software (such as Privacy Bird [8]) can check such specifications and decide if the interaction can proceed. Conflicting privacy settings might stop the interaction. For example if a user is not willing to



(a) Site Negotiation Profile (b) User Negotiation Profile (c) Possible Site-User Interactions

Figure 1. Two Negotiation Profiles and Possible Generated Interactions.

reveal her credit card, but the site requires it, the interaction cannot proceed. Negotiation could resolve such issues.

What makes negotiation possible is the existence of an underlying *utility function* that interacting parties aim to increase. Mechanisms that specify negotiable entities and utility functions are beyond the scope of this paper; however we envision protocols, similar to P3P, that allow such declarations. We assume each party declares a *negotiation profile* with negotiable entities and perceived utilities. Figure 1(a) is a possible negotiation profile of an e-shopping site showing negotiable entities (Bonus and Delivery Schedule), and perceived utilities against different user actions. Figure 1(b) is a corresponding user negotiation profile. The indicated numeric utility values could express monetary profit, gained popularity, satisfaction level, etc. We use the symbol ('-') to declare that a pair of entities is not negotiable. When interaction commences, profiles are *matched* forming the bi-matrices in Figure 1(c) which represent two potential interactions. The utility pair $(5, 30)$, for example, means that site gains a utility of 5 if it offered a discount in exchange of user's email, while user gains a utility of 30 if she revealed her email as a result of obtaining a discount. An *arbitration procedure* resolves each interaction in order to yield mutually beneficial outcomes. Such a procedure can be thought of as an 'expensive predicate' whose invocations need to be reduced.

1.1 Challenges and Motivation

The problems of multi-entities negotiation and optimal agenda were addressed by multi-agent systems [21]. It was shown that a large number of interactions could be initiated among agents to resolve negotiation issues either sequentially or simultaneously. However, proposed solutions do not address scalability in large-scale web transactions where a sheer amount of interactions could be initiated. Evaluating all possible interactions is infeasible because of the possibly complex negotiation mechanisms and additional constraints that interacting parties might have, e.g., budget or time constraints. Moreover, the outcome of some interaction might influence the outcomes of other interactions restructuring the problem space at each step. Discovering skyline and top-k arbitration solutions in these settings is challenging. The following examples highlight these challenges.

Example 1 *Consider a complex B2B transaction held between two companies, to trade products and services. Because of the wide varieties of products and services offered by each company, and the large set of business rules, one can think of a huge number of possible interactions that could be initiated. A trusted mediator entity, similar to an auctioneer in online auctions, could be selected by both companies to coordinate the negotiation. The mediator's task is to efficiently discover beneficial interactions to both companies. The outcome of each interaction influences the strategies of both companies, e.g. by deeming other interactions non-profitable or restricting their beneficial outcome space. Efficient ordering of the interactions to be processed is crucial to identify skyline in small number of steps.*

Example 2 *Consider a web user, U, surfing the web for the best e-shopping sites matching his privacy profile. U's agent could initiate interactions with many sites in order to maximize U's utility, which can be some function of privacy level and accessible services. An outcome of one interaction could be used by U's agent to redefine the space of beneficial outcomes from other interactions. Since in web-scale, a large number of sites could offer similar services to U, efficient processing of interactions to identify the skyline or top-k solutions is crucial in realizing this on-line scenario.*

The problems in previous examples connect with data management problems in uncertain data management [6, 9] and query trading [20]. However, our problem involves objects with non-traditional data characteristics which makes current skyline and bargaining methods not applicable. We summarize the challenges tackled in this paper as follows:

- *Non-traditional Characteristics:* Traditional skyline algorithms assume deterministic scores, while interaction outcomes are indeterministic. Arbitration procedures resolving interaction scores are considered expensive predicates whose usage needs to be reduced.

- *Dynamic Configuration:* Resolving an interaction offers additional 'knowledge' influencing the outcomes of solved/unsolved interactions (Section 3).

Moreover, processing all interactions collectively as one large interaction is hindered by the following challenges:

- *Negotiation Complexity:* Constructing and processing such a large interaction do not scale for large-scale web bargaining scenarios. Moreover, resolving an interaction becomes more complex with large uncertainty areas (Section 2).
- *Early Pruning:* Constructing a large interaction incurs unnecessary overhead to fully define all interactions, although non-beneficial interactions could be avoided without being completely defined (Section 3).
- *Autonomy:* Interactions should be treated as self-contained units since they represent distinct negotiation entities. Combining all interactions in one large interaction does not conform to this requirement.

These challenges motivate the need for new representation and computation models.

1.2 Contributions

We address the following data management problems in the context of large-scale web interaction and bargaining:

- Given an interaction framework, e.g. Figure 1, how to assign "importance" scores to possible interactions?
- For a large set of possible interactions, how to efficiently realize skyline or top-k bargaining solutions, while evaluating the minimum number of interactions?

We summarize our contributions as follows:

- We define *scored join results* in the context of web bargaining. We use game theory principles to *model* join results as *cooperative games*.
- We address skyline and top-k computation in web bargaining context, and show that interaction makes traditional algorithms prohibitively expensive.
- We conduct extensive experiments to evaluate our proposed techniques and show their superiority over traditional algorithms.

The remainder of this paper is organized as follows. Section 2 is necessary game theory background. Section 3 gives problem definition. Sections 4 and 5 present our proposed skyline and top-k techniques, respectively. Section 6 is our experimental study. Section 7 discusses related work. We conclude the paper in Section 8 with final remarks.

2 Background

In this section we provide necessary background material for the developments in this paper. We briefly present results from game theory. Interested reader is referred to the vast bibliography for a comprehensive treatment [16, 18].

Let M be a $m \times n$ bi-matrix with entries (a_{ij}, b_{ij}), $1 \leq i \leq m, 1 \leq j \leq n$. In our settings M can be derived by merging the *negotiation profiles* of two parties. Let P_1, P_2 be two parties (players). Once M is determined, a *game* is defined. P_1 and P_2 have m and n possible strategies, respectively. Once P_1 makes a choice, P_1's utilities for each possible choice of P_2 are known and they correspond to the a_{ij} values in the chosen row of the matrix (similarly for P_2). Each player makes a choice at *random*. The probabilities with which various strategies are chosen will probably be known to the other player, but the particular strategy chosen at a particular play of the game will not be known. The problem for each player, is to set probabilities in an optimal way. A *mixed* strategy for P_1 is an m-tuple $\vec{p}$ of probabilities such that $p_i \geq 0, 1 \leq i \leq m$ and $\sum_{i=1}^{m} p_i = 1$. Similarly for P_2 we have an n-tuple $\vec{q}$ of probabilities such that $q_j \geq 0, 1 \leq j \leq n$ and $\sum_{j=1}^{n} q_j = 1$.

The expected utility (payoff) for P_1 due to mixed strategies $\vec{p}, \vec{q}$ is $\pi_1(\vec{p}, \vec{q}) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_i q_j a_{ij}$ and the expected utility for P_2 is $\pi_2(\vec{p}, \vec{q}) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_i q_j b_{ij}$. Each player can compute a pessimistic utility estimate, called the *maximin* value, by assuming that the other player will act so as to minimize players's utility.

It is possible that P_1 and P_2 make binding agreements about strategies to play. A *joint* strategy is a probability matrix $P = (p_{ij})$. Thus, $p_{ij} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n$ and $\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} = 1$. A joint strategy assigns a probability to each strategy pair. The expected utility (payoff) of P_1 due to joint strategy P is $\pi_1(P) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} a_{ij}$, and similarly for P_2 using b_{ij}. Since players cooperate, the resulting game is *cooperative*. The *cooperative* payoff region is the set $\{(\pi_1(P), \pi_2(P))$: P is a joint strategy$\}$. Cooperative payoff regions are closed bounded convex sets with vertices among bi-matrix entries [16]. Cooperative game players make an agreement about which joint strategy to adopt based on: (1) the payoff pair of adopted strategy is not dominated; and (2) each player gain is at least the same as maximin value. These considerations lead to the following definitions:

Definition 1 ***Bargaining Set.*** *The bargaining set of a two player cooperative game is the set of all payoff pairs (u, v) such that $u \geq v_1, v \geq v_2$, where v_1, v_2 are the maximin values.*

Definition 2 ***Pareto-optimal Solutions.*** *The pareto-optimal solutions of a two player game is the set of all pay-*

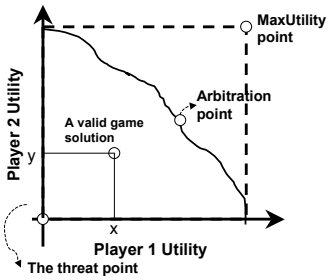


Figure 2. Example game.

off pairs (u, v) *such that there is no other payoff pair* $(\acute{u}, \acute{v})$ *with* $\acute{u} > u$ *and* $\acute{v} > v$.

Nash bargaining model [17, 16] established the basis for an arbitration procedure to decide which payoff pair should be agreed on. Based on a payoff region P and a status quo point $(u_0, v_0) \in P$, an arbitration procedure returns a mutually beneficial payoff pair (u^*, v^*). The status quo point is usually the pair of maximin values. Nash axioms state: (a) [Individual Rationality] $u^* \geq u_0$ and $v^* \geq v_0$, (b) [Pareto Optimality] (u^*, v^*) is pareto-optimal and (c) [Feasibility] $(u^*, v^*) \in P$. The main result of Nash [17] is that there exists a unique arbitration procedure satisfying these axioms.

Given an $m \times n$ bi-matrix M, we can approximate its payoff region while enclosing the arbitration pair. An upper bound for the payoff region is derived by $m_1 = max_{i,j} a_{ij} 1 \leq i \leq m, 1 \leq j \leq n$ and $m_2 = max_{i,j} b_{ij} 1 \leq i \leq m, 1 \leq j \leq n$; notice that $u^* \leq m_1$ and $v^* \leq m2$. Also $v_1 \leq u^*$ and $v_2 \leq v^*$. Given any bi-matrix M, the region enclosed by the rectangle $((v_1, v_2), (m_1, m_2))$ contains the arbitration pair. We refer to this rectangle as the game *bounding rectangle*, and to (m_1, m_2) and (v_1, v_2) as the $MaxUtility\ point$ and $threat\ point$ respectively (Figure 2). Whenever we obtain the arbitration point of a game we say that we *solved* or (*played*) that game.

Identifying arbitration pair is an expensive computational task [16, 18]. We do not discuss the details here, however we emphasize that the complexity of the arbitration procedure is proportional to the game uncertainty area as it involves a linear program over all possible solutions.

Notice that cooperation guarantees more profitable outcomes than competition. Consider Figure 1(c), the competitive Nash equilibria [17] are (10,10) and (15,15) for the two depicted games, while cooperative equilibria are (20,20) and (30,30). Solving games cooperatively requires, however, *complete* utility information. There are several other theoretical methodologies where *incomplete* utility information is available, e.g., utilities are specified as probability distributions. The discussion in this paper assumes cooperative games with complete information. Our future work involves studying other problem variants where games with incomplete information might be adopted.

3 Problem Definition

We view the interaction between two negotiation profiles (one from each party) as a "join" process that produces a set of "cooperative games". Each game encapsulates a set of corresponding policies from each party. The solutions of cooperative games define the scores of join results. Our main goal is to find the set of games with maximum or non-dominated scores. More formally, given a collection of N games let S be a set of rectangles obtained by deriving the $threat\ point$ and the $MaxUtility\ point$ of each game (each rectangle encloses an arbitration pair), the goal is to find the pareto-optimal set of arbitration pairs in S. For each arbitration pair, (u_i^*, v_i^*), in the pareto-optimal set, there exists no arbitration pair (u_j^*, v_j^*) with $u_j^* > u_i^*$ and $v_j^* > v_i^*$.

Exact arbitration pair is only known when solving the game. Since we approximate each game by a rectangle, we aim to utilize such approximation to minimize the *total number of solved games* to identify the pareto-optimal set. There are two considerations affecting this approach:

- *Game Dominance:* Solving a game might make other games uninteresting. Let (u_i^*, v_i^*) denote the arbitration pair of game i. Let (m_1^j, m_2^j) refer to the $MaxUtility\ point$ of game j. Game i dominates game j if $u_i^* > m_1^j$ and $v_i^* > m_2^j$. When game i dominates game j, game j immediately becomes uninteresting, since its arbitration point is not pareto-optimal. Consider Figure 3. Games $g1$, $g2$, and $g4$ can be pruned before processing any of the 10 games, since the $MaxUtility$ points of these games are dominated by the $threat\ point$ of other games. Games $g6$ and $g7$ can be safely pruned after solving game $g9$; since $g9$ solution dominates the $MaxUtility\ point$ of $g6$ and $g7$, making them uninteresting.

- *Game Clipping:* The knowledge gained by the solution of some games may force us to reconsider previous solutions. Consider Figure 3 again. After solving $g8$, we know that we can gain a utility of u_{g8}^*, and therefore the $threat\ point$ of $g5$ needs to be redefined. This essentially redefines the entire game and may result in a new arbitration point for $g5$. The solution of $g8$ has a similar effect on $g7$. As a result, rectangles corresponding to solved or yet unsolved games are *clipped*.

It is clear from Figure 3 that the order at which we solve games significantly affects the total number of games need to be played; solving $g7$ before $g9$ causes redundant playing of one extra game ($g7$). We distinguish two problem settings: (1) *no-interaction*, where only game dominance relationships are considered; and (2) *allowing interaction*, where both game dominance and clipping are considered.

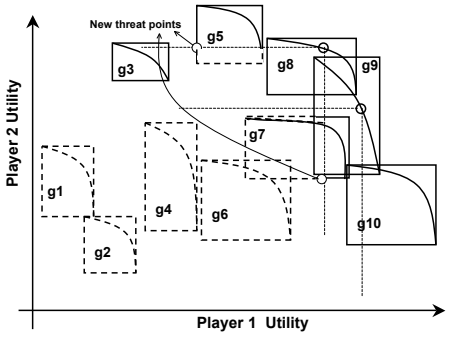


Figure 3. Example game set.

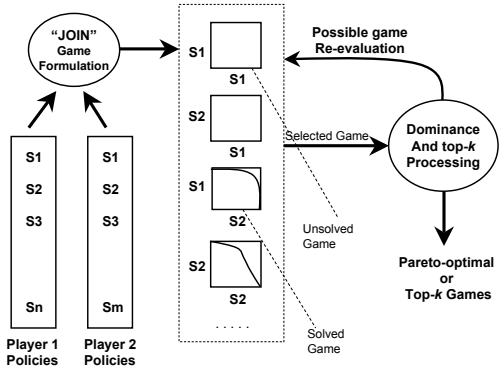


Figure 4. Proposed problem formulation.

Solving a game in the latter case might change other games by clipping their solution space necessitating "replay" of these games. We are now ready to state the main problem addressed by this paper: *Find a complete sequence of game playing to minimize the number of need-to-solve games while finding pareto-optimal solutions. A sequence is "complete" if no games need to be played after playing the last game in the sequence.*

Figure 4 is a straw-man design of proposed problem formulation. The join process formulates cooperative games based on input profiles. Formulating games requires only partial information about players utilities, namely $threat\ point$ and $MaxUtility\ point$ without actually considering the complete payoff matrix. To illustrate consider Example 2: User's agent need not have prior detailed information about gained utilities from each site. In fact the agent might never contact some sites for detailed information if it is known that their best possible outcome can never make it to the skyline or top–k results.

Formulated games are presented as input to our proposed dominance and top-k processing algorithms to select the next game to play. When a game is selected for playing, its payoff matrix needs to be fully defined, by contacting negotiating parties to obtain detailed utility information, in order to evaluate that game. Evaluating a certain game might lead to re-evaluation of other games. Note that our problem definition assumes the existence of a mediator (a trusted server in case of one-to-one interactions, or user's agent in one-to-many interactions) with knowledge about the utilities of negotiating parties in each possible interaction. In this first treatment of the problem, we decided to adopt such well-defined settings. Our future work in this area will involve more elaborate models where utilities are not fully exposed for privacy reasons as indicated in Section 2.

4 Finding the Pareto-optimal Solutions

In this section, we address the problem introduced in Section 3 in its simplest setting, where game solution does not affect other games definitions. In the following sections, we show how to generalize to the much harder scenarios, where game interaction is allowed. We show that no-interaction assumption simplifies the problem significantly, while allowing interaction causes exponential explosion in the search space and makes the simple techniques of Section 4.1 prohibitively expensive.

4.1 The Simple No-interaction Scenario

In the no-interaction scenario, solving any of the individual games reduces a game g to a single point (u_g^*, v_g^*), the arbitration point of g. A naïve approach is to solve all games, then apply a skyline algorithm [5, 19] to obtain the pareto-optimal game solutions. Solving all games can be prohibitively expensive when the number of games is large and the individual games are arbitrarily complex. Significant savings can be achieved by continuously pruning (eliminating) games that cannot contribute to final answer.

Our goal is to devise techniques to decide the best processing order of games to minimize the total number of games solved. Unfortunately, the optimal order cannot be devised before knowing the exact solution of each of the games. We propose game ranking criteria that aim at minimizing the number of games solved by ordering the games with respect to their "pruning power". Estimating the pruning power of a game before solving that game requires approximating the game solution (u_g^*, v_g^*).

We introduce two approximations for (u_g^*, v_g^*): (1) *Optimistic Approximation*, where (u_g^*, v_g^*) is approximated by the $MaxUtility\ point$; and (2) *Conservative Approximation*, where (u_g^*, v_g^*) is approximated by the middle point on the diagonal line between the two corners of the game bounding rectangle other than the $threat\ point$ and the $MaxUtility\ point$. The second approximation is more conservative, since game solution will be on the frontier of the cooperative payoff region (cf. Figure 2). Having an approximate game solution, $(\hat{u_g^*}, \hat{v_g^*})$, we introduce two heuristics to estimate game pruning power:

- *Pruning area*: Let o be the minimum utility point for the two players across all games. The rectangular area

between o and $(\hat{u_g^*}, \hat{v_g^*})$ represents the *Pruning Area* of g. Intuitively, the larger the pruning area, the higher the probability that solving g prunes more games.

- *Number of pruned games*: The pruning power of g is represented by the number of games fully dominated by $(\hat{u_g^*}, \hat{v_g^*})$ in the game set.

Ranking games by the pruning area is *context independent* and assumes the games are distributed uniformly in the game space. The heuristic is easy to compute since game rank depends only on the game bounding rectangle. In contrast, ranking games by the number of pruned games is *context-aware*; computing game rank requires information about all other games in the space. We obtain this information by indexing games rectangles using an R-Tree index in our implementation. In the experimental evaluation in Section 6, we compare the performance of the two approaches.

The general procedure to find pareto-optimal solutions of a set of non-interacting games is as follows: (1) estimate the solution of each game according to one of the two approximations, and use it to estimate game pruning power; (2) sort games in a descending pruning power order; (3) solve games in that order, and for each solved game, prune all uninteresting solved/unsolved games.

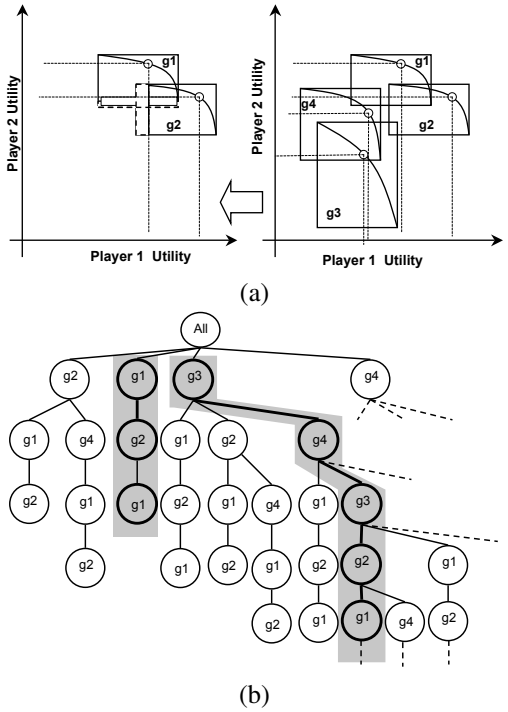


Figure 5. Game interaction and the search space.

4.2 Search Space with Interaction

The search space to identify optimal game sequence is a space of all valid complete sequences. We view this space as a search tree, where each node represents a game, and the first level consists of all games. The children of a node g are all possible games to be played after solving g. A leaf tree node l represents a terminal game; no further games need to be played. After solving each game, the game set may change by pruning or redefining other games according to the interaction model in Section 3. Each root-to-leaf tree path is a valid complete sequence, and our goal is to find the minimum length path.

For example, consider the games and the corresponding search tree in Figure 5. We concentrate on the two shaded paths. The first path starts by $g1$. Since $g1$ solution dominates $g3$ and $g4$, both are eliminated from the game set. The only possible game to play after $g1$ is a clipped version of $g2$, since the arbitration point of $g1$ dominates part of $g2$. Solving $g2$ redefines $g1$, and hence $g1$ needs to be resolved. The sequence is complete after solving $g1$ for the second time. In fact, the sequence $(g1, g2, g1)$ is one of the shortest sequences in this game setting. Now consider the other shaded path that starts by $g3$. The solution of $g3$ does not fully dominate any other game, hence $g1$, $g2$ and $g4$ are all possible children of $g3$. Solving $g4$ next, redefines $g3$, making it a possible child of $g4$. The path continues by solving $g2$, then $g1$. The path is not complete since $g1$ affects back $g2$ necessitating solving $g2$ again.

4.3 Identifying the Shortest Path

We introduce a branch-and-bound exhaustive search algorithm to explore the search tree and to find the shortest path from the root to a leaf node. The algorithm adopts a depth-first traversal pruning all search paths of length larger than the current shortest complete sequence.

Algorithm 1 describes the branch-and-bound algorithm. The algorithm starts by generating a search tree node with initial game configuration, and then moves down the tree, in a depth-first traversal, expanding each node by generating children for each game. When a node is reached such that its solution does not modify the game set and no other games need to be played, a complete sequence is obtained. Each node saves game configuration that was produced by playing the node's game. Note that the algorithm does not generate the complete search space; this is achieved by pruning paths that will not contribute to the shortest sequence.

As illustrated by Figure 5, node traversal order greatly affects pruning power. Picking a child that can prune most of the remaining games leads to a short complete sequence early in the search process, allowing more aggressive pruning of the search space. Algorithm 1 adopts one of our heuristics (discussed in more details later) to choose the

Algorithm 1 Guided Depth-First Search (DFS)

create $root$ node with all the games in the node's game set and mark it visited.
$node \leftarrow root$
$ShortestPath \leftarrow null$
while $node \neq null$ **do**
  if $ShortestPath \neq null$ and length of the path to $node$ $\geq$ length of $ShortestPath$ **then**
    $node \leftarrow$ parent of $node$
    $continue$ {path pruned}
  end if
  if $node$ is not visited **then**
    solve the $node$'s game and mark $node$ as *visited*
    redefine affected games in the $node$'s game set
    create a child node for each unsolved game
  end if
  $node \leftarrow$ an unvisited child of $node$ based on heuristics
  if $node = null$ **then** {complete sequence found}
    if path length to $node <$ length of $ShortestPath$ **then** {new shortest path found}
      $ShortestPath \leftarrow$ path from $root$ to $node$
    end if
    $node \leftarrow$ parent of $node$
    $continue$
  end if
end while
return $ShortestPath$

Algorithm 2 Finding a Good Sequence

initialize Q a priority queue of all unsolved games
initialize $S \leftarrow \{\}$ a set of all solved games
for each game g **do**
  compute $(\hat{u_g^*}, \hat{v_g^*})$
  compute $score_g$ according to one the two estimates of the pruning power of g
  insert g in Q with $score_g$
end for
while Q is not empty **do**
  $g \leftarrow Q.top$ and remove g from Q
  solve g to get (u_g^*, v_g^*)
  add g to S
  remove all fully dominated games from S
  remove all fully dominated games from Q
  for each game l affected by (u_g^*, v_g^*) **do**
    redefine l
    if $l \in S$ **then**
      remove l from S
      compute $(\hat{u_l^*}, \hat{v_l^*})$ and $score_l$
      insert l in Q with $score_l$
    end if
  end for
end while

most promising child. For example, consider the search tree in Figure 5(b) that corresponds to the game set in Figure 5(a). Search starts at the root by creating a child node with each game in the root game set $\{g1, g2, g3, g4\}$. The algorithm chooses $g2$ to solve first, leading to a complete sequence $(g2, g1, g2)$. The algorithm uses the length of the shortest complete sequence (3 in this case) to prune paths of length 3 and longer. Had the algorithm started with $g3$ first, longer complete sequence will be reached first, and hence less pruning is achieved.

Branch-and-bound algorithm is guaranteed to find a (or all) shortest complete sequences. However, the total number of solved games is much larger than the path length. Note that we cannot use the solution of some node's game when the same game is encountered at a different node because of different game configurations of the two nodes. In practice, we would like to minimize the overall number of played games until pareto-optimal solutions are obtained. That is, we aim to identify the first complete sequence in the search tree as a final answer (not for pruning). Hence, ordering the traversal of children, based on heuristics, is crucial in making the first complete sequence as short as possible.

4.4 Finding a "good" Sequence

Since many games are redefined when a game is solved, counting the number of pruned games by current game solution does not fully reflect the pruning power of that game. We need to factor in the number of games that get "affected" or redefined. Hence, we modify the second estimate of the pruning power of a game g described in Section 4.1 as follows: *Number of affected games*; the pruning power of g is represented by the number of games that are either fully or "partially" dominated by $(\hat{u_g^*}, \hat{v_g^*})$. The *Pruning Area* estimate remains unchanged since it is context-independent (referred to here as *Clipping Area*). Our heuristics evaluate pruning power after solving each game, and use it to score games. Next game to play is the game with the current highest score. We modify the general procedure in Section 4.1 to consider continuously changing game set.

Algorithm 2 maintains a priority queue, Q, of all unsolved games, and a set S of solved games. Initially all games are inserted in Q in descending pruning power, which is estimated according to one of two approximations: number of affected games, or clipping area. After solving game g, retrieved from Q top, fully dominated games are eliminated and affected games are redefined. Then, g is inserted in S, and all affected games in S are removed and reinserted in Q again. The algorithm terminates when Q is empty, where S contains the pareto-optimal solutions.

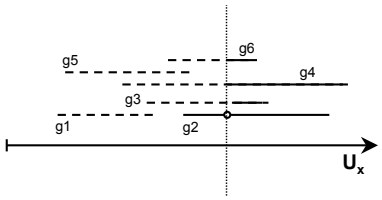


Figure 6. On-to-many game interaction.

4.5 One-to-Many Interactions

In this setting, two parties P_1, P_2 are involved, one defining a single negotiation profile and the other defining multiple profiles (refer back to Example 2). Our interest is to find the game (or top-k games) that maximize the utility/payoff of P_1 among all games that can be formed with the profiles of P_2. We would like to find the best game(s) while evaluating the minimum number of games in the game space. In this section, we address the problem of finding the best game. We address retrieving the top-k games in Section 5.

The problem can be viewed as a projection of all games on the P_1 utility axis. Each unplayed game is represented as an interval from $(min_g^{(x)}, max_g^{(x)})$. The game solution lies in the interval and can be obtained by playing the game cooperatively with the corresponding partner. We show that simpler versions of the techniques described earlier can be applied to solve this one-to-many problem.

The No-interaction Scenario As described in Section 4.1, in this scenario only full-dominance is allowed, i.e., a game is pruned without playing iff the solution of another game dominates its $MaxUtility\ point$. A naïve approach is to solve all games and sort them by P_1 utility in the arbitration point. Pruning can be also conducted on-line; pruning all fully dominated games after solving each game. Straightforward modifications to heuristics in Section 4.1 can be applied for more aggressive pruning.

Allowing Interaction In this scenario, a game solution can dominate part of another game solution space. Figure 6 shows game configuration projected on P_1's utility axis.

In Figure 6, game $g1$ was pruned before playing any game since the minimum utility of $g2$ dominates $g1$'s maximum profit. After solving $g2$ further pruning can be achieved by eliminating $g5$. Solving $g2$ also affects the definition of $g4$ and $g6$ by clipping part of their solution space.

5 Identifying Top-k Games

In previous sections, the output of our algorithms is a *set* of non-dominated (pareto-optimal) solutions. Choosing one of these solutions is probably application dependent. A related query is to get the top-k game solutions that maximize an application-defined function, $\mathcal{F}$ on players payoffs, e.g., the sum of the payoffs. Although related, pareto-optimal solutions are not guaranteed to contain the top-k solutions (only the top-1). In this section, we are interested in retrieving k games with the highest scores, where score is calculated by applying a function $\mathcal{F}$ on game solution. Top-k as defined is not applicable when game interaction is allowed, since pruning games is part of the problem definition. The semantics of top-k in this context is not clear; the game set changes by pruning and redefining games.

Algorithm 3 Get Next Top-k Game

```
1:  while Q is not empty do
2:      g ← Q.top and remove g from Q
3:      T ← F̄_g of Q.top
4:      if g is solved OR F̲_g ≥ T then
5:          return g and break;
6:      else
7:          solve g to get (u*_g, v*_g)
8:          F_g ← F(u*, v*)
9:          insert g in Q with F_g
10:     end if
11: end while
```

Note that $\mathcal{F}$ does not affect how games are solved, rather imposes a ranking on solutions. Retrieving top-k games incrementally is of interest in many cases. For example, in a space with one game g dominating the whole space, the two players might want to keep "alternative" games to be played in case of failure to implement g. A top-k mechanism allows to *incrementally* retrieve the next-best game.

Consider a function $\mathcal{F}(u_x, u_y)$ on players utility and a set of unsolved games. Let score of game g, $score_g$, be the result of evaluating $\mathcal{F}$ on (u_g^*, v_g^*). Hence, $score_g = \mathcal{F}(u_g^*, v_g^*)$. For a game g, let $\overline{\mathcal{F}_g}$ and $\underline{\mathcal{F}_g}$ be an upper-bound and a lower-bound of $score_g$, respectively. Algorithm 3 incrementally retrieves the top-k games with respect to $\mathcal{F}$. The algorithm assumes a priority queue, Q, of all games ranked on their upper-bound score $\overline{\mathcal{F}}$. When a game is solved, its score can be accurately calculated. Algorithm 3 reports (on each invocation) the next top-k game according to $\mathcal{F}$. Note that no pruning of games is performed, instead all games are kept ranked on their score.

It can be shown that if $\mathcal{F}$ is monotone, then applying $\mathcal{F}$ on $MaxUtility\ point$, gives a tight upper-bound and makes the aforementioned algorithms optimal in the number of solved games. Since $\overline{\mathcal{F}_g}$ is the tightest upper-bound that can obtained without solving a game or making a random guess; the optimality follows from the $\mathcal{A}^*$ search algorithm. We omit the proof details due to space constraints.

Algorithm 3 introduces further optimization, by allowing the top-k game to be reported without solving if it is guaranteed to score higher than all other games. The idea is to keep a lower bound on the game score (computed on

the $threat\ point$) and report a game when the lower-bound score is higher than the upper-bound score of all games.

6 Experiments

We experimentally evaluated our techniques and compared heuristics in different game settings. All experiments were run on a 3GHz Pentium IV PC with 1 GB of main memory and 40 GB of disk space, running Windows XP. We built a Java tool to manipulate games in a 2D space, track solution paths of different heuristics, and generate online statistics. We used two synthesized datasets: (1) *DSU1*: Uniformly distributed game sets in 2D space; and (2) *DSU2*: Games are clustered around the line between the two points $(0, Max_u)$ and $(Max_v, 0)$, where Max_u and Max_v are the maximum players utility, respectively. *DSU2* corresponds to the case where games are competitive with no clear winners. We evaluate four heuristics in different settings:

- *Max (Avg) Clipping* : Game pruning power is based on the area between the origin and the game's $MaxUtility$ (center) point.
- *Loosely (Tightly) Affected Games* : Game pruning power is the number of games partially or fully inside the area between the origin and the game's $MaxUtility$ (center) point.

Our performance metrics are: (1) the length of complete sequence; (2) the total running time; and (3) the number of unnecessary game solvings.

6.1 No-Interaction Scenario

This experiment measures complete sequence length for each heuristic in a *No Interaction* scenario. Figure 7 compares heuristics to optimal path length (obtained from Algorithm 1). Because of huge search space, we conducted the experiment only for small number of games from DSU1. Figure 8 compares heuristics to a *no order* strategy for a large number of games in DSU2. Significant saving in sequence length is achieved by ordering games by their pruning power. Among different heuristics, the *Tightly Affected Games* performed the best, achieving up to 50% reduction in sequence length compared to solving games randomly.

6.2 Interaction Scenario

This experiment measures the complete sequence length of each heuristic in the *Interaction* scenario. Figures 9 and 10 show the sequence length for each heuristic in different game sets drawn from DSU1. Note that the DFS algorithm could not be used for comparison in large data sets because of the explosive growth in interactions as games are solved; this makes it very costly to explore the whole search space and get the global optimal solution. Therefore, we compare the heuristics among each other for large sets. Our experiments show similar results for DSU2. The *Tightly Affected Games* heuristic generates the shortest sequences compared to other ordering heuristics. This saving in sequence length is due to the context-awareness of the *Affected Games* heuristic as discussed in Section 4.1. The *Loosely Affected Games* heuristic shows a relatively worse behavior compared to the *Tightly Affected Games* in large data sets. This is explained by the large variation in game size which leads the *Loosely Affected Games* heuristic to false estimates whenever the solution point of a game lies far from its $MaxUtility\ point$.

Figures 11 and 12 show the running times of different heuristics for DSU1 and DSU2 respectively. Notice that starting from 20,000 games, the time consumed by the two *Affected Games* heuristics largely exceeds the time consumed by the *Clipping Area* heuristics. This is attributed primarily to the extra processing required by the *Affected Games* heuristic to count the number of potential game interactions after each step. This processing is not needed for the *Clipping Area* heuristics since they evaluate each game based on the game's own (local) properties only.

6.3 Heuristics Effectiveness

This experiment evaluates the *effectiveness* of sequences generated by different heuristics. The *effectiveness* of a complete sequence *S* is the ratio of the number of *effective* solved games to $|S|$. A game g is *effective* if the solution of g modifies game set, or it is in the pareto-optimal solutions. The intuition behind this definition is that playing an *ineffective* game is primarily due to wrong estimates of game pruning power. Figures 13 shows heuristics effectiveness for DSU2 (results for DSU1 are omitted for space limitations). In DSU1, the average effectiveness of all heuristics is around 0.7, while it is lower for DSU2 (0.65 for *Tightly Affected Games* and below 0.5 for others). This is attributed to DSU1 nature, where games appear arbitrarily in the space which involves many effective games. In contrast, in DSU2, it is harder to frequently find a game that affects other games after the first few steps. The *Tightly Affected Games* heuristic did not suffer because it takes into account the possible number of affected games in each step.

6.4 Limited Game Interaction Scenario

We study a limited interaction scenario where a limit, $MaxInfleunce$, is imposed on the number of times a game is redefined. This represents the case where solution space of some games is fixed after lengthy bargaining process. Increasing *MaxInfluence* results in two contradicting phe-

nomena. Figure 14 shows the trade-off for a configuration of 1000 games: By increasing *MaxInfleuence*, the sequence length starts to decrease as more clipping and pruning is allowed. Further increase of *MaxInfleuence* does not achieve shorter paths as no more games are pruned. Sequence length increases again as competitive games cause multiple game replays through mutual interaction. *Tightly Affected Games* heuristic is less prone to this trade-off by taking into account game configuration in each step.

6.5 Top-k Bargaining Games

This experiment evaluates the efficiency of Algorithm 3 by measuring how many games are solved to report the top-k games. Figure 15 shows that the number of solved games is almost linear with respect to k. The Processing time of Algorithm 3 also increases linearly with k.

7 Related Work

Game theory is very rich [16, 18]. Several game models have been studied in the literature including cooperative and non-cooperative games. Numerous game theory applications exist in diverse computer science fields.

A large body of work addresses skyline computation, first studied in [14]. Recent skyline algorithms include a Block-nested-loop algorithm [5], a Sort-filter-skyline algorithm [7], and several index-based algorithms [5, 22, 19, 12]. The algorithm in [19] is a branch-and-bound algorithm to progressively output skyline points based on R-tree index, and guarantees the minimum I/O cost. The nature of our problem (as explained in Section 3), makes it substantially different from traditional skyline computation.

8 Conclusions

We presented a novel problem of identifying pareto-optimal and top-k bargaining solutions in large-scale web interaction. We introduced a basic problem setting in this context, and several of its variants. We showed that brute-force techniques are prohibitively expensive and proposed algorithms and heuristics to efficiently solve different problem variants. Our work raises several interesting questions for further study; identifying and analyzing problems complexity, providing hardness results or polynomial time algorithms, and studying the applicability of more elaborate models are important future directions. We believe that our basic methodology and algorithms can be of use to other settings as well (e.g., uncertain data management in which uncertainty reduction is expensive). We plan to further explore such connections as part of our future work.

References

[1] Inspire: http://interneg.org/inspire.
[2] Simplens: http://mis.concordia.ca/simplens.
[3] Smartsettle: http://www.oneaccordinc.com.
[4] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *TOIS*, 21(2), 2003.
[5] S. Borzsonyi, D. Kossmann, , and K. Stocker. The skyline operator. In *ICDE*, 2001.
[6] N. Chaudhry, J. Moyne, and E. Rundensteiner. An extended database design methodology for uncertain data management. *Inf. Sci. Inf. Comput. Sci.*, 121(1-2):83–112, 1999.
[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
[8] L. F. Cranor. *Web Privacy with P3P*. O'Reily, 2001.
[9] V. de Almeida and R. Hartmut. Supporting uncertainty in moving objects in network databases. In *GIS*, pages 31–40, 2005.
[10] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: a system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, 2001.
[11] I. F. Ilyas, R. Shah, W. G. Aref, J. S. Vitter, and A. K. Elmagarmid. Rank-aware query optimization. In *SIGMOD*, 2004.
[12] D. Kossmann, F. Ramsak, and S. Rost. Shooting starts in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.
[13] G. Koutrika and Y. Ioannidis. Personalized queries under a generalized preference model.
[14] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *JACM, 22(4):469–476*, 1975.
[15] C. Li, K. Chang, I. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.
[16] P. Morris. *Introduction to Game Theory*. Spinger, 1991.
[17] J. F. Nash. The Bargaining Problem. *Econometrica*, 1950.
[18] G. Owen. *Game Theory*. Academic Press, 1984.
[19] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.
[20] F. Pentaris and Y. E. Ioannidis. Distributed query optimization by query trading. In *EDBT*, pages 532–550, 2004.
[21] M. W. S. S. Fatima and N. R. Jenning. Optimal agendas for multi-issue negotiation. In *AAMAS*, 2003.
[22] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.
[23] J. H. W. Jin and M. Ester. Mining thick skylines over large databases. In *PKDD*, 2004.

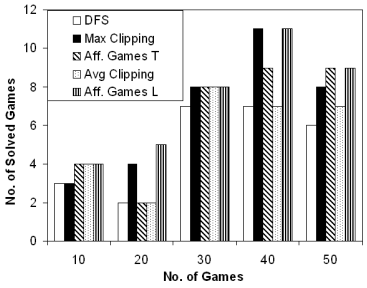


Figure 7. No Interaction (DSU1)

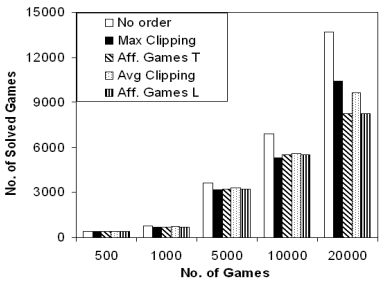


Figure 8. No Interaction (DSU2)

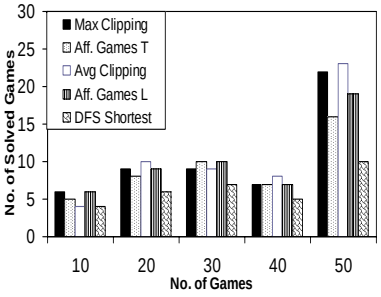


Figure 9. Interaction (DSU1 small sets)

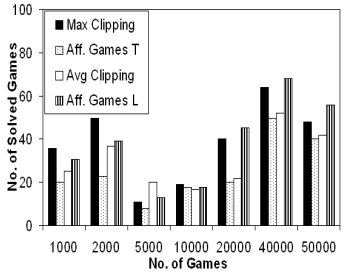


Figure 10. Interaction (DSU1 large sets)

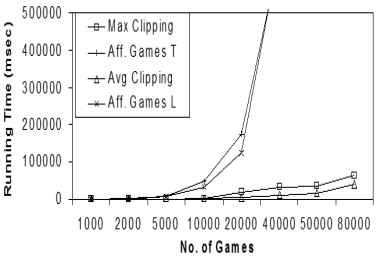


Figure 11. Running Times (DSU1)

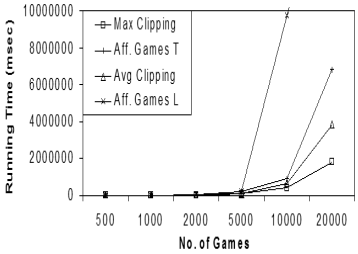


Figure 12. Running Times (DSU2)

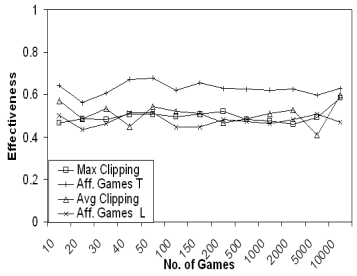


Figure 13. Effectiveness (DSU2)

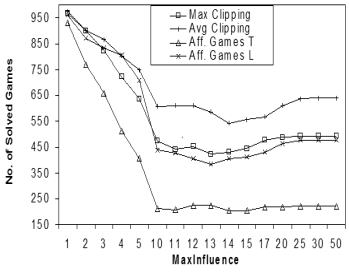


Figure 14. *MaxInfluence* (1000 Games)

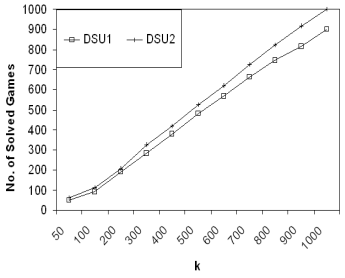


Figure 15. Solved Games in Top-k Selection Algorithm